Package 'Rgretl'

March 18, 2018

Type Package

Title Interface to 'gretlcli'

Version 0.2.2

Date 2018-03-17

Author Oleh Komashko

Maintainer Oleh Komashko <oleg_komashko@ukr.net>

Depends R (>= 3.4.0)

Imports compiler, lubridate, png, xml2

LazyData yes

SystemRequirements gretl (>= 2017c), gretlcli

Description An interface to 'GNU gretl': running 'gretl' scripts from, estimating econometric models with backward passing of model results, opening 'gretl' data files (.gdt). 'gretl' can be downloaded from <http://gretl.sourceforge.net>. This package could make life on introductory/intermediate econometrics courses much easier: full battery of the required regression diagnostics, including White's heteroskedasticity test, restricted ols estimation, advanced weak instrument test after iv estimation, very convenient dealing with lagged variables in models, standard case treatment in unit root tests, vector auto-regressions, and vector error correction models. Datasets for 8 popular econometrics textbooks can be installed into 'gretl' from its server. All datasets can be easily imported using this package.

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2018-03-18 15:24:53 UTC

R topics documented:

`Rgretl-package` *Interface to* gretlcli

Description

An interface to *GNU gretl*: running *gretl* scripts from R, estimating econometric models with backward passing of model results, opening *gretl* data files '`.gdt`'. *gretl* can be downloaded from `http://gretl.sourceforge.net`. This package could make life on introductory/intermediate econometrics courses much easier: *gretl* provides full battery of the required regression diagnostics, including White's heteroskedasticity test, restricted OLS estimation, advanced weak instrument test after iv estimation, very convenient dealing with lagged variables in econometric models, standard case treatment in unit root tests, vector autoregressions, and vector error correction models. Datasets for 8 popular econometrics textbooks can be installed into *gretl* from its server. All datasets can be easily imported using this package.

Details

Package:	Rgretl
Type:	Package
Version:	0.2.2
Date:	2018-03-17
Depends:	R (>= 3.4.0)
SystemRequirements:	gretl (>= 2017c), gretlcli

Index of help topics:

```
Rgretl-package          Interface to _gretlcli_
datasets_info           extracting information on _gretl_ sample files
description             printing description of a data frame
find_sample             finding sample scripts which use a _gretl_
                        command specified by name
greene                  sample scrips from _gretl_ 'Greene' selection
gretl                   sample scrips from _gretl_ 'Gretl' selection
```

```
grmod                   Interface for _gretl_ estimation commands
merge_data              merging data frames and/or (multiple)
                        time-series objects as simple list
open_gdt                opening _gretl_ and _gretl_-supported data
                        files
ramanathan              sample scrips from _gretl_ 'Ramanathan'
                        selection
run_grcli               running gretl scripts
save_bin                saving data as _gretl_ data bases
save_gdt                saving data as _gretl_(-export) data files
show_sample             printing _gretl_ sample scripts
```

Getting started

For the package to work properly *gretl* should be installed. The installation instruction is in README file.

The first thing to *run* is

```
run_grcli("help help")
```

Yes! Run, run, help, help! Imagine, a hungry tiger is after you and there will be nothing for humdrum cramming. The output will show how to use *gretl* help system from a script or console, hence using `run_gretlcli`. In contrast with R *gretl* has commands and functions. A command syntax is similar to that of shell commands. Run
run_grcli("help")
run_grcli("help functions")

to obtain lists of commands and functions. Let's turn to a slighly longer script:
myfirstscript <-
'nulldata 50
eval 2 + 2
set seed 13
x1 = normal()
x2 = normal()
y = x1 + x2 + normal()
y = y > 0
printf "\\nThis is my first gretl model:\\n"
logit y 0 x1 x2
varlist --accessors'
run_grcli(myfirstscript)

It is convenient to render a part of the output:
This is my first gretl model:
Model 1: Logit, using observations 1-50
Dependent variable: y
Standard errors based on Hessian

	coefficient	std. error	z	slope
const	0.398073	0.360291	1.105	
x1	1.15444	0.523459	2.205	0.280427
x2	1.50154	0.521053	2.882	0.364742

...

model-related
$ess (scalar: 4.92415)
$T (scalar: 50)

Since `myfirstscript` is an R character vector rather than a genuine *gretl* script file it uses '\\n' instead of '\n' (as it should be in the native script) to indicate a new line in formatted printing. The single quote (') is used only inside strings in *gretl*. Hence if a user does not apply sohpisticated string manipulation, there exists one by one correspondence. Otherwise, some additional experimenting with '\"' may be needed. Instead of turning this subsection into a kind of textbook it should be sufficient to give a couple of hints. To understand the script above and its output one can proceed by running `run_grcli("help nulldata")`, `run_grcli("help eval")`, `run_grcli("help normal --func")`, etc. The output of varlist --accessors is an analogue of output of `names(obj)` and/or `names(summary(obj))` where `obj` is an estimated model object, e.g. an object of class `"glm"`. In *gretl* parlance dollar-prefixed objects are called *accessors*. Getting help on them follows the same pattern: run_grcli("help $ess").

A user can broadcast named variables backward to R:

```
logitres <- run_grcli(myfirstscript, output = list(series = c("y","$uhat"),
                matr = c("$coeff","$vcv","$rsq","$pd","$windows")))
names(logitres)
logitres$'$rsq'
class(logitres$y)
logitres$'$coeff'
logitres$'$vcv'
```

Just reproduce and look at the output. Note that both scalars and matrices included into `matr` component of `output`. It worth mentioning that accessors listed by varlist --accessors are present in the memory during *gretl* session, so they needn't to be included into `text` argument of `run_grcli` to be broadcasted to R. By contrast, in the example above the set of admissible 'non-dollar' series names includes only `x1`, `x2`, and `y`, since there are no other 'non-dollar' series names inside `myfirstscript`. If a data file is open, its series also can be broadcasted without mentioning them in the script:

```
logmoney <- run_grcli("open denmark.gdt -q", output = list(series = "LRM"))
head(logmoney$LRM).
```

We end the subsection by reproducing a simple trick to estimate the 'calling' time:

```
system.time(run_grcli('set stopwatch\n eval $stopwatch'))
```

Using *gretl* sample data files and sample scripts

Sample data files

If *gretl* is installed and is visible from inside R all *gretl* sample files are ready to be explored. Additional information is supplied by `Rgretl::datasets_info` and `Rgretl::description` functions.

Sample scripts

Three data files in **Rgretl** (`gretl.rdata`, `greene.rdata`, and `ramanathan.rdata`) contain character string data only. Its contents is formed by adopted versions of *gretl*-supplied `hansl` language sample scripts. In the first place, these scripts are easily searcheable and available from *gretl* GUI but not from *gretlcli*. Secon, since `run_grcli` deals with R character strings, and not with the original *hansl* scripts some minor changes are needed to compensate enclosing quotes. As a matter of fact, all changes appears to be inserting additional backslashes in several cases: (\) was substituted for (\\) to break long lines in code, (\n) – for (\\n) in formatted printing (`prinf`) `command`, (') was

substituted for (\') in comments and for matrix transposition when single quotes were used to enclose character string input to `Rgretl::run_grcli`, etc. All scripts can be easily run via `run_grcli` and all have been tested in the batch mode having produced a neat nearly 1MB-sized text log file. See also help entries for these data files for the further information.

Reading and saving data files

Reading

The `open_gdt` function can open data files of the following formats: '`*.gdt, *.gdtb`' (*gretl* data files), '`*.csv`', '`*.txt`', '`*.gnumeric`' (Gnumeric), '`*.ods`' (Open Document), '`*.xls, *.xlsx`' (Excel), '`*.dta`' (Stata), '`*.wf1`' (Eviews), '`*.sav`' (SPSS), '`*.xpt`' (SAS xport), '`*.dat`' (JMulTi). For example, one can run

```
denmark <- open_gdt("denmark")
```

```
# .gdt extension can be omitted in case (i) below
```

```
capm4 <- open_gdt("http://www.principlesofeconometrics.com/poe4/data/eviews/capm4.wf1")
```

We need not indicating path to a file in two cases:

(i) for *gretl* sample data files. The list of available files is at the bottom of README file. Additional sample files can be downloaded and installed via *gretl GUI*. README file contains installation instruction and the list of data files.

(ii) for files in *gretl* working directory (as extracted from *gretl* or by running `get_grwd`).

The output is a `data.frame` object. Curretntly, *time-series* structure is preserved (output data frame columns will be `ts` objects with the right attributes) only for yearly, quorterly, and monthly data. Otherwise, an additional column will be present for a user to recode it into time series structure by means of R. *Panel data* structure is easily recovered if time and unit indicators are present. When there are no visible time/unit identifiers a user can run

```
my.frame <- open_gdt("a_file.gdt")
```

```
time_n_id <- run_grcli('genr time',data = "a_file.gdt",output = list(series = c("time","$unit")))
```

```
my.frame$time <- time_n_id$time
```

```
my.frame$id <- time_n_id$'$unit'
```

Now `my.frame` is ready. A user can load her favourite R *package* for estimation of panel data models.

Another useful cheat sheet: running

```
run_grcli('help $datatype\n eval $datatype', data = "denmark")
```

will give
$datatype
Output: scalar
Returns an integer value representing the sort of dataset that is currently
loaded: 0 = no data; 1 = cross-sectional (undated) data; 2 = time-series
data; 3 = panel data.
2

Saving

To this end the package have two functions: `save_gtd` for saving data frames and/or (multiple) time-series objects as *gretl* data files (this should be useful mostly for *gretl*-oriented users); for a wider target group the most interisting feature is possibility to export data to Stata format; `tsave_bin`

saves lists with time series components as *gretl* binary data bases'`*.bin`'; the latter is supplied by a helper, `merge_data` for easy constructing lists with the appropriate structure to export, also this function supplies all series with unique names as required by *gretl* for saving its binary data bases. The best way to render panel data structure is providing R data frames with explicit unit and period identifiers before exporting.

Plotting

Simplified plotting

To this end *gretl* has `textplot` command which outputs simple plots in console. To see a working example execute

```
data(ramanathan)
cat(paste0(ramanathan$PS3.1,"\n"))
run_grcli(ramanathan$PS3.1)
```

True plots

gretl has several plotting commands: `boxplot`, `gnuplot`, `graphpg`, `hfplot`, `plot`, `qqplot`, `rmplot`, `scatters`. All commands above have `'--output=filename'` option, e.g. `'--output=myfile.pdf'`, or `'--output=display'`. Execute `run_grcli("help gnuplot")` to see the details on `'--output=filename'`. "Saving" options work as intended in **Rgretl**. At the other hand, `'--output=display'` option may output nothing on some systems if being called from **Rgretl**. To compensate this sad fact **Rgretl** has a special option for *gretl* plotting commands: `--output=#R`. With this option plots will be rendered on R standard graphic windows. A user can copy-paste and execute the code below to see how it works:

```
a_script <-
'nulldata 93
setobs 4 1995:1 --time-series
set seed 13
y = normal()
RandomWalk = cum(y)
x = normal()
gnuplot RandomWalk --time-series --with-lines --output=#R
qqplot y x --output=#R
'
run_grcli(a_script)
```

Note that `'--output=#R'` option is not intended to work with *gretl* `graphpg` command. Also, vector graphics is not retained currently. To insert publishing-quality plots in a tect editor use one of the "saving" options.

Estimation of econometric models

It is a wide practice inside R to have several alternatives to estimating certain models. From the autor's subjective point of view the most valuable for elementary/intermediate econometrics courses features include post-estimation menu (execute `run_grcli("help modtest")` and references therein), heteroscedasticity toolbox (execute `run_grcli("help hsk")`). *gretl* provides a large toolbox for very easy working with dynamic regressions. Also *gretl* provides full case treatment for unit root testing and cointegration analisys. Written on c commands for MLE estimation of models for limited and discrete dependent variable are very quick and reliable.

The `grmod` function in this version by default outputs a sizable list of model output similar to that of `lm`, `glm`, etc. Still, not all estimation commands are processed by `grmod`. In such cases `run_grcli`

provides an alternative; thought this way requires much more typing from a user. Some useful extensions are considered in README file.

Author(s)

Oleh Komashko

Maintainer: Oleh Komashko <oleg_komashko@ukr.net>

References

Cottrell, A., and Lucchetti, R. (2018) "Gretl User's Guide," http://ricardo.ecn.wfu.edu/pub//gretl/manual/en/gretl-guide.pdf

Cottrell, A., and Lucchetti, R. (2018) "Gretl Command Reference", http://ricardo.ecn.wfu.edu/pub/gretl/manual/PDF/gretl-guide.pdf

Cottrell, A., and Lucchetti, R. (2018) "A Hansl Primer", http://ricardo.ecn.wfu.edu/pub/gretl/manual/PDF/hansl-primer.pdf

---

`datasets_info` *extracting information on* gretl *sample files*

Description

Extracts information on *gretl* sample data files which are include into *gretl* installation. This information is ready available via *gretl* GUI menu but not via *gretlcli*. This function makes the use of **Rgretl** tmore independent. All possible output list the set of files which can be opened by `open_gdt` without indicating *neither* path no extension.

Usage

```
datasets_info(directory = NULL, prn = NULL)
# dirs = datasets_info()
# gretl = datasets_info(dirs[1])
```

Arguments

`directory`	`void` or character vector, see Examples.
`prn`	integer vector: determines what subset of files to print a short description. By default prints all available information; if it is set to zero printing is suppressed, see Examples.

Value

character vector

Author(s)

Oleh Komashko

Examples

```
dirs = datasets_info()
gretl = datasets_info(dirs[1],1:2)
head(gretl)
```

`description` *printing description of a data frame*

Description

Prints "description" attribute of an object if such attribute exists. Otherwise returns `NULL`. It is intended to be applied to data frames created by `open_gdt`.

Usage

```
description(x)
# denmark <- open_gdt("denmark")
# description(denmark)
```

Arguments

`x`	any object, supposedly a data frame created by `open_gdt`. If `x` has `"description"` attribute the function prints its contents; otherwise returns `NULL`.

Value

character vector

Author(s)

Oleh Komashko

Examples

```
## Not run:
denmark <- open_gdt("denmark")
description(denmark)

## End(Not run)
```

`find_sample` *finding sample scripts which use a* gretl *command specified by name*

Description

Findis sample scripts which use a *gretl* command specified by name. Sample scripts which are present in **Rgretl** as data are searched.

Usage

```
find_sample(cmd, dir = NULL)
# find_sample("panel")
# find_sample("wls","ramanathan")
```

Arguments

`cmd`	*gretl* command as character string, execute `run_grcli("help")` for the list of valid *gretl* commands.
`dir`	*gretl* character string, can be `"gretl"`, `"greene"`, or `"ramanathan"`; specifies a name of scripts collection to search; if missing, all three are searched.

Value

character vector): contains the name(s) of found sample scripts as components of corresponding data sets.

Author(s)

Oleh Komashko

Examples

```
find_sample("panel")
cat(paste0(gretl$penngrow,"\n"))
run_grcli(gretl$penngrow)
```

greene *sample scrips from* gretl *'Greene' selection*

Description

sample scrips from *gretl* 'Greene' selection translated as R character vectors; they can be executed via `run_grcli`.

Usage

```
greene
# names(greene)
# show_sample(greene$greene7.8)
# run_grcli(greene$greene7.8)
```

Format

List containing 13 sample scripts (= character vectors) which can be executed via `run_grcli`. See entry for `ramanathan` data set.

Source

gretl sample scripts

gretl *sample scrips from* gretl *'Gretl' selection*

Description

sample scrips from *gretl* 'Gretl selection translated as R character vectors; they can be executed via `run_grcli`.

Usage

```
gretl
# names(gretl)
# show_sample(gretl$reprobit)
# run_grcli(gret$reprobit)
```

Format

List containing 40 sample scripts (= character vectors) which can be executed via `run_grcli`. See entry for `ramanathan` data set.

Source

gretl sample scripts

grmod *Interface for* gretl *estimation commands*

Description

The function provides interface for gretl estimation commands using R data frames, or gretl native data files.

Usage

```
grmod(mdl,data,top = character(0),bottom = character(0),
      input = list(matr = list(),char = character()),
      output=list(matr=character(), char=character(),
                  series = character()))

# With gretl native data file; outputs coeff, vcov, etc
# mod <- grmod("arima 0 1 1; 0 1 1; lg --nc", "bjg.gdt")
# names(mod)

# With AirPassengers from datasets package
# grmod("arima 0 1 1; 0 1 1; log(AirPassengers) --nc",AirPassengers)
```

Arguments

mdl	quoted gretl estimation command, e.g. "ols y const x", or command block, see also Examples.
data	can be one of two types: (i) an R data.frame, ts, or mts object; (ii) quoted name of gretl data file, or name of a file gretl can open as data file.
top	quoted gretl script: it may contain e.g. code for preliminary data transformations, or unrestricted ols estimation before restrict block.
bottom	quoted *gretl* script: designed for post-estimation, e.g "modtest --autocorr".
input	determines additional optional input, for example, the matrix of initial values for optimization (typically of likelihood maximization for arima, mle, nls, etc).
output	determines additional optional components of the output list, e.g. p-value for some diagnostic model test, see Value below and entry for output argument of run_grcli.

Details

In the current version of the package, *all gretl* estimation commands except for midasreg are processed.

Panel data

If data argument is a *gretl* data file with panel settings nothing special is needed. Otherwise, the first line of of top argument should be setobs id time_id --panel-vars (use actual names!).

Command blocks

Some of *gretl* estimation commands use several lines of code, e.g. `restrict` (with `--full` option), `system`, `mle`, `gmm`, etc.

`restrict`: `mdl` argument should consist of several lines encompassing all `restrict` block, see Examples below

Other command blocks: `mdl` argument should contain only the last line of the block (typically the first "word" of the last line is "end", e.g. `end mle`); all previous lines shold be included into `top` argument.

Value

a list containing imported output of correspondent *gretl* command output; if available `$model` *bundle* is captured, otherwise available dollar-prefixed accessors are transmitted. Scalars, series, column and row vectors are rendered as R numeric vectors. Matrices of other shapes are rendered as R matrix objects. As always a user can execute run_grcli("help $vcv"), etc.

Author(s)

Oleh Komashko

References

Cottrell, A., and Lucchetti, R. (2018) "Gretl User's Guide," http://ricardo.ecn.wfu.edu/pub/gretl/manual/en/gretl-guide.pdf

Examples

```
set.seed(13)
dfr = data.frame(y = rnorm(20), x = rnorm(20))
grmod("ols y 0 x --simple-print",dfr)
grmod("ols diff(LRY) 0 diff(LRM)","denmark.gdt")


## Not run:
## command block example
# unrestricted linear regression
tp <- "ols diff(LRY) 0 diff(LRM) IBO -q"
# restrict block code:
md <-
"restrict --full
b[2] + b[3] = 0.3
end restrict"
# estimation
mod1 <- grmod(md,"denmark.gdt",top=tp)
names(mod1)

## prelimanary data manipulation example
pr <-
"list DEMOG = age educ female black married
list INCOME = finc25 finc35 finc50 finc75 finc100 finc101"
```

```
# estimate ordered probit
grmod("probit pctstck choice DEMOG INCOME wealth89 prftshr",
      "pension.gdt",top=pr,type="ordered")


## End(Not run)
```

---

merge_data *merging data frames and/or (multiple) time-series objects as simple list*

Description

Merges data frames and/or (multiple) time-series objects as simple list, with retainig time series structure (if any). If there were duplicated names, some components are renamed for all output list elements to have unique names. Can be used for creating datasets where individual series can have different lenghts and frequencies. Also it is used for data preparation to create *gretl* data bases.

Usage

```
merge_data(...)
# merge_data(airmiles,AirPassengers)
```

Arguments

...	R data objects, each can be `data.frame`, `ts`, or `mts` object.

Value

list

Author(s)

Oleh Komashko

Examples

```
all_data <- merge_data(airmiles,AirPassengers)
head(all_data[1])
head(all_data[2])
```

open_gdt *opening* gretl *and* gretl-*supported data files*

Description

The function can open data files of the following formats: '*.gdt, *.gdtb' (*gretl* data files), '*.csv', '*.txt', '*.gnumeric' (Gnumeric), '*.ods' (Open Document), '*.xls, *.xlsx' (Excel), '*.dta' (Stata), '*.wf1' (Eviews), '*.sav' (SPSS), '*.xpt' (SAS xport), '*.dat' (JMulTi).

Usage

```
open_gdt(fpath, mkstruct = TRUE, info = TRUE)
# open_gdt("denmark.gdt")
```

Arguments

fpath	quoted name of a data file; the rules of adding the full path are the same as in gretl; can open data files of all gretl-supported formats: Stata, E-views, SPSS, SAS, etc. Path to a file is not needed for *gretl* sample files and for files in *gretl* working directory
mkstruct	logical; whether to preserve time series structure; currently only yearly, quarteerly, monthly, weekly and dayly data are supported: in these cases the output data frame columns will be ts objects with 'right' time-series attributes; otherwise the output data.frame contains character-valued column named "obs" which can be translated into time structure by a user.
info	logical; whether to import data description (if any); if info = TRUE (the default) the output data frame will have additional "description" attribute : a character vector containing data set information. It can be printed in intelligible form by description function.

Value

data.frame if info = TRUE it will have "description" attribute.

Author(s)

Oleh Komashko

Examples

```
denmark = open_gdt("denmark.gdt")
head(denmark)
description(denmark)

## Not run:
```

```
gold = open_gdt("http://www.principlesofeconometrics.com/poe4/data/stata/gold.dta")
description(gold)
## End(Not run)
```

---

ramanathan *sample scrips from* gretl *'Ramanathan' selection*

Description

sample scrips from *gretl* 'Ramanathan' selection translated as R character vectors; they can be executed via `run_grcli`.

Usage

```
ramanathan
# names(ramanathan)
# show_sample(ramanathan$PS3.1)
# run_grcli(ramanathan$PS3.1)
```

Format

List containing 73 sample scripts (= character vectors) which can be executed via `run_grcli`. Use `names(ramanathan)` to see the list of scripts; use `show_sample(ramanathan[[i]])` to print a script in intelligible format; use `run_grcli(ramanathan[[i]])` to execute it (i in 1:73). *gretl* sample scripts are readily available via GUI but not via command line interface. Hence, this set of scripts make the use of **Rgretl** more self-sufficient. A printing of `show_sample(ramanathan[[i]])` looks exactly the same as the original *hansl* script (*hansl* is the name of *gretl* built-in language). Since *hansl* scripts were translated as R character vectors, modifications were made in using quotes and escapes: single quotes (') were prepended by backslashes (\') in comments and and in code properly where it serves as matrix transposition sign, and backslashes (\) were substituted for double backslashes (\\) for breaking long lines of code (it is very similar to typing this fragment of '`ramanathan.Rd`' file: I had to type triple backslash above for a reader would see the double one). This is exactly what to be done if a user wishes copy-paste printing into R script editor and create an R character string enclosed by single quotes (say, to use it as a template for her own code).

Source

gretl sample scripts

run_grcli *running gretl scripts*

Description

The function runs gretl scripts from inside R. Currently it outputs a list with components specified by output argument.

Usage

```
run_grcli(text,
          input = list(matr = list(), char = character()),
          output=list(matr=character(), char=character(),
                      series = character()),
          data=NULL)
# run_grcli(text) # only prints
# run_grcli(text, data = data_frame)
# run_grcli(text, data = "gret_file.gdt")
```

Arguments

text	gretl script formed as a character vector, see Examples.

input	determines additional components of input: scalars, vectors, matrices, and character strings:
	`matr = list(name1 = value1,name2 = value2,...)`
	`char = c(name1 = value1,name2 = value2,...)`
	For 'matr' values can be numeric vectors, or matrices. For example, `run_grcli('print x',input = lis` will print `x = 0.78539816`. Numeric vectors will be translated as [*gretl*] column matrices. If x is explicitly created in R as a matrix object its dimentions will be preserved.
	'char' values are one-element character vectors.

output	determines components of the output list: it should contain names of gretl script objects (in the text argument) as character vectors; matr can contain names of *gretl* scalars and matrices; char can contain names of *gretl* string objects; series can contain names of *gretl* series. Series are broadcasted back to R as numeric vectors.
data	a name (without quotes) of an R data.frame, ts, or mts object; a name of *gretl* (or *gretl*-supported, e.g. Stata, Eviews, SAS, SPSS, etc) data file as character string.

Value

A list with components specified by output argument. Output components could be numeric atomic vectors, matrices, or character vectors; gretl scalars, row and column matrices, and series are converted to atomic vectors.

Author(s)

Oleh Komashko

Examples

```
# The differences with the native gretl script are:
# (i) name of character vector (gr.script <- here)
# (ii) quotes: for make hansl code a character string;
#      quotes inside script should be escaped, e.g.
#      text <- "eval \"hello, word\""
gr.script <-
"open denmark.gdt -q
ols diff(LRY) 0 diff(LRM) IBO --simple-print
restrict --silent
b[2] + b[3] = 0.3
end restrict
pv = $pvalue
if pv > 0.05
restrict --full
b[2] + b[3] = 0.3
end restrict
set warnings off
modtest --white-nocross
endif"
run_grcli(gr.script)

Rz = run_grcli('print w v x\n eval x**y\n z = mexp(y)\n print z',
           input = list(matr = list(w = 1,v = 1:4, x= matrix(1:2,1),
           y = matrix(rep(1,9),3))),
           output = list(matr = "z"))
Rz$z

## Not run:
# After gretl is properly installed one can
# safely copy-past and run this example

scr <-
"nulldata 10
set seed 13
series y = normal()
matrix x = seq(1,7)
matrix z = mshape({y},5,2)
string str = \"Hello, user!\"
print str"
ou <- run_grcli(scr,output = list(matr=c("x","z"),char="str",series = "y"))
ou
## End(Not run)
```

---

save_bin *saving data as* gretl *data bases*

Description

Saves data in the form of a list containing time-series objects of different lengths and frequencies as gretl data bases.

Usage

```
save_bin(f_name, data_list, overwrite = FALSE, select = NULL)
```

Arguments

f_name	the name of file to be created or changed, as character vector; *gretl* data bases have 'bin' extension, which will be appended by default. The default path is current R working directory.
data_list	non-branched list (e.g. output of merge_data); its components must be ts objects; the set of admissible frequencies is determined by those, supported by *gretl* data bases: monthly, quortery, yearly; if data_list contains series with another frequencies they will not not saved and warning(s) will be produced.
overwrite	logical; save_bin is essentially a wrapper over *gertl* store "filename" --database [--overwrite] command; setting overwrite = TRUE switches on the corresponding option; execute run_grcli("help store") for details.
select	integer vector, must be a subset of 1:length(data_list); data_list could consist of dozens of thousands series, so a user may want to recode not all series.

Value

void

Author(s)

Oleh Komashko

Examples

```
## Not run:
save_bin("airmix ", merge_data(AirPassengers,airmiles))

## End(Not run)
```

---

save_gdt *saving data as* gretl*(-export) data files*

Description

Saves data frames and/or (multiple) time-series objects as gretl data files, or in any the *gretl*-supported data files formats e.g. Stata, Octave, PcGive data files.

Usage

```
save_gdt(fname, ...)
# save_gdt(c("AirPassengers.gdt", "AirPassengers.dta"), AirPassengers, AirPassengers.gdt)
# save_gdt("AirPassengers.dat --jmulti", AirPassengers)
# save_gdt("AirPassengers.m --gnu-octave", AirPassengers)
# save_gdt("AirPassengers.dat --dat", AirPassengers)
```

Arguments

fname	character vector consisting of names of data files to be created; supported extensions are 'gdt', 'gdtb' (both *gretl*), 'txt', 'csv', 'asc', 'dta' (Stata), 'dat' (JMulTI, adding `--jmulty` is needed, see Usage above), 'm' (Octave, adding `--gnu-octave` is needed, see Usage above), 'dat' (PcGive, adding `--dat` is needed, see Usage above). Omitted extension is the same as '.gdt'. The default path is current R working directory. In '.gdt' and '.gdtb' files dayly, weekly, monthly, quarterly, and yearly series will retain dates and frequencies.
...	R data objects, each can be `data.frame`, `ts`, or `mts` object, the number of arguments in ... **must** coinside with the length of `fname`.

Value

void

Author(s)

Oleh Komashko

Examples

```
## Not run:
save_gdt("AirPassengers", AirPassengers) # saving as '.gdt'

path <- paste0(get_grwd(),"AirPassengers")
save_gdt(path, AirPassengers) # saving to gretl working directory

## End(Not run)
```

---

show_sample *printing* gretl *sample scripts*

Description

Prints *gretl* sample scripts. Outputs script contents (invisible). It is convenient when the function argument is a name, rather then a script itself.

Usage

```
show_sample(x)
# show_sample("gretl$nile")
# show_sample(gretl$nile)
```

Arguments

x	character vector; a *gretl* sample script, or its name.

Value

character vector

Author(s)

Oleh Komashko

Examples

```
mles <- find_sample("mle")
mle1 <- show_sample(mles[1])
## Not run:
run_grcli(mle1)

## End(Not run)
```

Index