

Package: ritest (via r-universe)

March 22, 2025

Type Package

Title Randomisation Inference Testing

Version 0.1.0

Maintainer <grantmcd@uoregon.edu>

Description An experimental port of the `ritest` Stata routine by Simon Heß. Fast and user-friendly. Aims to support a variety of model classes once it is fully baked.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

URL <https://github.com/grantmcdermott/ritest>,
<http://grantmcdermott.com/ritest>

BugReports <https://github.com/grantmcdermott/ritest/issues>

Imports utils, stats, grDevices, graphics, parallel, parallelly,
data.table, pbapply, fixest (>= 0.10.0), multcomp (>= 1.4-18),
generics, cli

Suggests tinytest, ggplot2, broom, covr, knitr, rmarkdown,
modelssummary

RoxygenNote 7.1.2

Depends R (>= 2.10)

VignetteBuilder knitr

Repository <https://grantmcdermott.r-universe.dev>

RemoteUrl <https://github.com/grantmcdermott/ritest>

RemoteRef HEAD

RemoteSha 5ac25bb56b93e3ce6eaaef8056df13fca295cb6a

Contents

colombia	2
glance.ritest	3
plot.ritest	4
print.ritest	6
ritest	7
tidy.ritest	11
Index	13

colombia	<i>Vendor data from an RCT conducted in Bogota, Colombia.</i>
----------	---

Description

This dataset is a preliminary version of the data described in Iacovone and McKenzie (forthcoming). It conforms to an earlier blog post written by one of the authors (McKenzie, 2017) and contains data collected during a randomized controlled trial on supply chains among fresh produce vendors in Bogota, Colombia.

Usage

```
colombia
```

Format

A data frame with 3336 rows and 8 variables:

b_block business block (geographic identifier)

b_pair randomly assigned treatment pairs (plus one triplet)

b_treat treatment status (1 = yes, 2 = no)

dayscorab number of days that week requiring visits to Corabastos central market

b_dayscorab baseline of the previous variable

miss_b_dayscorab dummy for missing baseline information (1 = yes, 2 = no)

round2, round3 survey round dummies

Details

The RCT studies the impact of purchase aggregation by many microenterprises (here: fruit and vegetable vendors), which enables a reduction in costly individual visits to a large central market. My thanks to David McKenzie for sharing the data.

References

McKenzie, D. (2017) "Finally, a way to do easy randomization inference in Stata!", Development Impact (World Bank blog). <https://blogs.worldbank.org/impactevaluations/finally-way-do-easy-randomizati>

Iacovone, L. and McKenzie, D. (forthcoming) "Shortening Supply Chains: Experimental Evidence from Fruit and Vegetable Vendors in Bogota", Economic Development and Cultural Change. <https://doi.org/10.1086/714050>.

glance.ritest	<i>Glance at an 'ritest' object</i>
---------------	-------------------------------------

Description

Glance at an 'ritest' object

Usage

```
## S3 method for class 'ritest'
glance(x, ...)
```

Arguments

x	An object produced by the 'ritest' function.
...	Additional arguments. Currently ignored.

Value

A data frame of goodness-of-fit statistics that conforms to the 'broom' package specifications.

Examples

```
est = lm(yield ~ N + P + K, data = npk)
est_ri = ritest(est, 'N', reps = 1e3, seed = 1234L, pcores = 2L)
tidy(est_ri)
glance(est_ri)
```

plot.ritest

A plot method for ritest objects

Description

Nice plots of your ritest objects.

Usage

```
## S3 method for class 'ritest'
plot(
  x,
  type = c("density", "hist"),
  highlight = c("lines", "fill", "both", "none"),
  show_parm = FALSE,
  breaks = "auto",
  family = NULL,
  ...
)
```

Arguments

x	An ritest object.
type	Character. What type of plot do you want?
highlight	Character. How do you want to highlight the H0 rejection regions in the distribution tails?
show_parm	Logical. Should we highlight the parametric H0 rejection regions too?
breaks	Character. Histogram plot only. What type of breaks do you want? The default method creates more breaks than the standard R behaviour. You can revert to the latter by selecting NULL.
family	Character. The font family. Defaults to 'HersheySans' instead of R's normal Arial plotting font.
...	Other plot arguments. Currently ignored.

Examples

```
#
## Example 1: Basic functionality
#

# First estimate a simple linear regression on the base 'npk' dataset. For
# this first example, we won't worry about strata or clusters, or other
# experimental design complications.
est = lm(yield ~ N + P + K, data = npk)

# Conduct RI on the 'N' (i.e. nitrogen) coefficient. We'll do 1,000
```

```

# simulations and, just for illustration, limit the number of parallel cores
# to 2 (default is half of the available cores). The 'verbose = TRUE'
# argument simply prints the results upon completion, including the original
# regression model summary.
est_ri = ritest(est, 'N', reps = 1e3, seed = 1234L, verbose = TRUE)

# Result: The RI rejection rate (0.021) is very similar to the parametric
# p-value (0.019).

# We can plot the results and various options are available to customise the appearance.
plot(est_ri)
plot(est_ri, type = 'hist')
# etc

# Aside: By default, ritest() conducts a standard two-sided test against a
# sharp null hypothesis of zero. You can specify other null hypotheses as
# part of the 'resampvar' string argument. For example, a (left) one-sided
# test...
plot(ritest(est, 'N<=0', reps = 1e3, seed = 1234L, pcores = 2L))
# ... or, null values different from zero.
plot(ritest(est, 'N=2', reps = 1e3, seed = 1234L, pcores = 2L))

#
## Example 2: Real-life example
#

# Now that we've seen the basic functionality, here's a more realistic RI
# example using data from a randomized control trial conducted in Colombia.
# More details on the dataset -- kindly provided by the study authors -- can
# be found in the accompanying helpfile ("colombia"). The most important
# thing to note is that we need to control for the stratified (aka "blocked")
# and clustered experimental design.

data("colombia")

# We'll use the fixest package to estimate our parametric regression model,
# specifying the strata (here: treatment-control pairs) as fixed-effects and
# clustering the standard errors by location (here: city blocks).
library(fixest)
co_est = feols(dayscorab ~ b_treat + b_dayscorab + miss_b_dayscorab |
               b_pair + round2 + round3,
               vcov = ~b_block, data = colombia)
co_est

# Run RI on the 'b_treat' variable, specifying the strata and clusters.
co_ri = ritest(co_est, 'b_treat', strata='b_pair', cluster='b_block',
               reps=1e3, seed=123L)
co_ri
plot(co_ri, type = 'hist', highlight = 'fill')

# This time, the RI rejection rate (0.11) is noticeably higher than the
# parametric p-value (0.024) from the regression model.

```

print.ritest	<i>A print method for ritest objects</i>
--------------	--

Description

Printed display of ritest objects. Tries to mimic the display of the equivalent Stata routine.

Usage

```
## S3 method for class 'ritest'
print(x, verbose = FALSE, ...)
```

Arguments

x	An ritest object.
verbose	Logical. Should we display the original model summary too? Default is FALSE.
...	Currently ignored.

Examples

```
#
## Example 1: Basic functionality
#

# First estimate a simple linear regression on the base 'npk' dataset. For
# this first example, we won't worry about strata or clusters, or other
# experimental design complications.
est = lm(yield ~ N + P + K, data = npk)

# Conduct RI on the 'N' (i.e. nitrogen) coefficient. We'll do 1,000
# simulations and, just for illustration, limit the number of parallel cores
# to 2 (default is half of the available cores). The 'verbose = TRUE'
# argument simply prints the results upon completion, including the original
# regression model summary.
est_ri = ritest(est, 'N', reps = 1e3, seed = 1234L, verbose = TRUE)

# Result: The RI rejection rate (0.021) is very similar to the parametric
# p-value (0.019).

# We can plot the results and various options are available to customise the appearance.
plot(est_ri)
plot(est_ri, type = 'hist')
# etc

# Aside: By default, ritest() conducts a standard two-sided test against a
# sharp null hypothesis of zero. You can specify other null hypotheses as
# part of the 'resampvar' string argument. For example, a (left) one-sided
```

```

# test...
plot(ritest(est, 'N<=0', reps = 1e3, seed = 1234L, pcores = 2L))
# ... or, null values different from zero.
plot(ritest(est, 'N=2', reps = 1e3, seed = 1234L, pcores = 2L))

#
## Example 2: Real-life example
#

# Now that we've seen the basic functionality, here's a more realistic RI
# example using data from a randomized control trial conducted in Colombia.
# More details on the dataset -- kindly provided by the study authors -- can
# be found in the accompanying helpfile ("?colombia"). The most important
# thing to note is that we need to control for the stratified (aka "blocked")
# and clustered experimental design.

data("colombia")

# We'll use the fixest package to estimate our parametric regression model,
# specifying the strata (here: treatment-control pairs) as fixed-effects and
# clustering the standard errors by location (here: city blocks).
library(fixest)
co_est = feols(dayscorab ~ b_treat + b_dayscorab + miss_b_dayscorab |
               b_pair + round2 + round3,
               vcov = ~b_block, data = colombia)
co_est

# Run RI on the 'b_treat' variable, specifying the strata and clusters.
co_ri = ritest(co_est, 'b_treat', strata='b_pair', cluster='b_block',
               reps=1e3, seed=123L)
co_ri
plot(co_ri, type = 'hist', highlight = 'fill')

# This time, the RI rejection rate (0.11) is noticeably higher than the
# parametric p-value (0.024) from the regression model.

```

ritest

Perform randomization inference on a model object

Description

Perform randomization inference (RI) testing on a model object, e.g. a coefficient from a linear regression model. It tries to mimic the ‘-ritest-’ Stata routine (Heß, 2017) in its design and functionality. The package is quite experimental and only a subset of this functionality is currently supported. However, it does appear to be significantly faster.

Usage

```

ritest(
  object,
  resampvar,
  reps = 100,
  strata = NULL,
  cluster = NULL,
  level = 0.95,
  parallel = TRUE,
  ptype = c("auto", "fork", "psock"),
  pcores = NULL,
  stack = NULL,
  stack_lim = 1L,
  seed = NULL,
  pb = FALSE,
  verbose = FALSE,
  ...
)

```

Arguments

object	Model object containing the ‘resampvar’ variable. At present, only ‘stats::lm’ and ‘fixest::feols’ models are supported.
resampvar	Character or one-sided formula. The variable (coefficient) that you want to perform RI on. By default, the RI procedure will conduct a standard two-sided test against a sharp null hypothesis of zero (i.e. $H_0: \text{resampvar} = 0$). Other null hypotheses may be specified as part of a character string. These must take the form of the resampled variable, followed by an (in)equality sign, and then a number. For example ‘x>=0’ or ‘x<=0’ would yield respective one-sided tests against the zero null. Similarly, you can test against values other than zero (e.g. ‘x=1’ or ‘x>=1’). However, note that multiple multiple comparison tests (e.g. ‘x1-x2=0’) are not yet supported. See Examples.
reps	Integer. The number of repetitions (permutation draws) in the RI simulation. Default is 100, but you probably want more than that. Young (2019) finds that rejection rates stabilise at around 2,000 draws.
strata	Character or one-sided formula. Permute ‘resampvar’ within strata (AKA blocks)? See Details and Examples below.
cluster	Character or one-sided formula. Keep ‘resampvar’ constant within clusters? See Details and Examples below.
level	Numeric. The desired confidence level. Default if 0.95.
parallel	Logical. Should the permuted fits be executed in parallel? Default is TRUE, with additional options being passed to the ‘ptype’ and ‘pcores’ arguments.
ptype	Character. What type of parallel strategy should be used? The default behaviour on Linux and Mac is parallel forking ("fork"), while on Windows it will revert to parallel sockets ("psock"). Note that forking is more efficient, but unavailable on Windows.

pcores	Integer. How many parallel cores should be used? If none is provided, then it will default to half of the total available CPU cores on the user's machine.
stack	Logical. Should the permuted data be stacked in memory all at once, rather than being recalculated during each iteration? Stacking takes advantage of vectorisation and is thus more efficient. (It also helps to ensure reproducibility when the results are being generated in parallel — see the note on random-number generation below.) But it does require additional memory. If no explicit choice is provided, then the function will automatically stack as long this implies an additional memory overhead less than the 'stack_lim' argument. Note that stacking is only relevant if at least one of 'strata' or 'cluster' are defined.
stack_lim	Numeric. What is the memory limit (in gigabytes) for determining whether the permuted data should be stacked in memory ahead of time? Default is 1 GB. You probably want to increase this if you are working with a large dataset and have lots of RAM.
seed	Integer. Random seed for reproducible results. Note that the choice of parallel behaviour can alter results even when using the same seed. See the Note on random number generation below.
pb	Logical. Display a progress bar? Default is FALSE. Progress bars can add a surprising amount of computational overhead to iterative functions. I've therefore set the number of updating steps — i.e. via the 'nout' argument of [pbapply::pboptions()] — to 5, which should limit this kind of overhead in the event that a user invokes the 'pb = TRUE' argument.
verbose	Logical. Display the underlying model 'object' summary and 'ritest' return value? Default is 'FALSE'.
...	Additional arguments. Currently ignored.

Details

This function is experimental and functionality is still quite limited. Albeit, that it does support the most likely use case for RI on a regression model, i.e. permutation testing of a coefficient value. Present limitations include: only 'lm' and 'fixest::feols' model objects are supported; only one permutation (RI) test is allowed; and only one strata and/or cluster variable, respectively, can be supplied. I hope to resolve these limitations as time permits.

Value

An list object of class 'ritest'. Default print and plotting methods are supported.

Note

The use of parallelism in computation introduces some well-known complications with respect to random number generation (RNG). See the vignette accompanying the 'parallel' package for more discussion. The 'ritest()' function adopts various best practices to facilitate reliably reproducible results, irrespective of the choice of parallel strategy. These include the use of "L'Ecuyer-CMRG" RNG kind and the default behaviour of "stacking" the permuted results in memory before passing them on to the fitting stage of the randomization inference routine (where the fitting would still be done in parallel). To briefly dwell on the latter, note that this stacking behaviour is very similar to

— I nearly wrote "parallels" — the approach adopted by 'boot' and other packages that trade off RNG with parallelism and reproducibility. In the phrasing of the 'parallel' vignette: "One way to avoid any difficulties is (where possible) to do all the randomization in the master process" (p. 5).

The upshot is that running 'ritest()' under a given set of arguments should generally yield the same, reproducible results regardless of when or where they were run. This should be true even if users turn the default parallel behaviour of the function off or back on, or whether they change the 'ptype' argument to "psock" or "fork" and back again. However, it cannot be guaranteed under every scenario. Perhaps the most obvious case is when 'ritest()' is run without any declared strata or clusters. (Reason: Stacking in this simple case is turned off because it yields no efficiency gains.) In this case, the RNG stream will be sensitive to the `_number_` of available cores on a user's computer. Two parallel cores will yield a slightly different result than four cores, or six cores, etc. Of course, users can always ensure perfect reproducibility by explicitly defining the number of required cores in the 'ritest()' call via the 'pcores' argument. Stepping back, the focus on reproducible exactitude rather misses the point in an exercise like randomization inference testing. Much more important is running enough permutation trials so that your rejection rates are stable and any minor differences due to different RNG seeds are moot. Remember, the ultimate goal of inference (and research) isn't simply to generate reproducible results under a very specific set of circumstances. Rather, it is to generate consistent insights that hold even under varying circumstances.

References

Simon Heß (2017). *Robust Randomization inference with Stata: A guide and software*, The Stata Journal, 17, Number 3, pp. 630–651

Alwyn Young (2019). *Channeling Fisher: Randomization Tests and the Statistical Insignificance of Seemingly Significant Experimental Results*, The Quarterly Journal of Economics, 134, Issue 2, pp. 557–598

See Also

[print.ritest()], [plot.ritest()]

Examples

```
#
## Example 1: Basic functionality
#

# First estimate a simple linear regression on the base 'npk' dataset. For
# this first example, we won't worry about strata or clusters, or other
# experimental design complications.
est = lm(yield ~ N + P + K, data = npk)

# Conduct RI on the 'N' (i.e. nitrogen) coefficient. We'll do 1,000
# simulations and, just for illustration, limit the number of parallel cores
# to 2 (default is half of the available cores). The 'verbose = TRUE'
# argument simply prints the results upon completion, including the original
# regression model summary.
est_ri = ritest(est, 'N', reps = 1e3, seed = 1234L, verbose = TRUE)

# Result: The RI rejection rate (0.021) is very similar to the parametric
```

```

# p-value (0.019).

# We can plot the results and various options are available to customise the appearance.
plot(est_ri)
plot(est_ri, type = 'hist')
# etc

# Aside: By default, ritest() conducts a standard two-sided test against a
# sharp null hypothesis of zero. You can specify other null hypotheses as
# part of the 'resampvar' string argument. For example, a (left) one-sided
# test...
plot(ritest(est, 'N<=0', reps = 1e3, seed = 1234L, pcores = 2L))
# ... or, null values different from zero.
plot(ritest(est, 'N=2', reps = 1e3, seed = 1234L, pcores = 2L))

#
## Example 2: Real-life example
#

# Now that we've seen the basic functionality, here's a more realistic RI
# example using data from a randomized control trial conducted in Colombia.
# More details on the dataset -- kindly provided by the study authors -- can
# be found in the accompanying helpfile ("colombia"). The most important
# thing to note is that we need to control for the stratified (aka "blocked")
# and clustered experimental design.

data("colombia")

# We'll use the fixest package to estimate our parametric regression model,
# specifying the strata (here: treatment-control pairs) as fixed-effects and
# clustering the standard errors by location (here: city blocks).
library(fixest)
co_est = feols(dayscorab ~ b_treat + b_dayscorab + miss_b_dayscorab |
               b_pair + round2 + round3,
               vcov = ~b_block, data = colombia)
co_est

# Run RI on the 'b_treat' variable, specifying the strata and clusters.
co_ri = ritest(co_est, 'b_treat', strata='b_pair', cluster='b_block',
               reps=1e3, seed=123L)
co_ri
plot(co_ri, type = 'hist', highlight = 'fill')

# This time, the RI rejection rate (0.11) is noticeably higher than the
# parametric p-value (0.024) from the regression model.

```

Description

Tidy an 'ritest' object

Usage

```
## S3 method for class 'ritest'  
tidy(x, conf.int = TRUE, ...)
```

Arguments

x	An object produced by the 'ritest' function.
conf.int	Logical indicating whether or not to include a confidence interval.
...	Additional arguments. Currently ignored.

Value

A data frame of summary statistics that conforms to the 'broom' package specifications.

Examples

```
est = lm(yield ~ N + P + K, data = npk)  
est_ri = ritest(est, 'N', reps = 1e3, seed = 1234L, pcores = 2L)  
tidy(est_ri)  
glance(est_ri)
```

Index

* **datasets**

colombia, [2](#)

colombia, [2](#)

glance.ritest, [3](#)

plot.ritest, [4](#)

print.ritest, [6](#)

ritest, [7](#)

tidy.ritest, [11](#)