ritest – compute p values for Monte Carlo permutation tests, allowing for arbitrary randomization procedures

Syntax

```
ritest permvar exp_list [, options] : command
```

Options	*Description*
Main	
reps(#)	perform # permutations, default is reps(100)
left|right	compute one-sided p-values; default is two-sided
Automatic permutation	
strata(varlist)	permute *permvar* within strata
cluster(varlist)	keep *permvar* constant within cluster
File-based permutation	
permfile(filename)	take permutations of *permvar* from stata data-file *filename*, with variables named [permvarname]1, [permvarname]2, [permvarname]3, …
permmatchvar(varlist)	merge permutations in *permfile* with the data using these variables (1:1 or m:1)
Program-based permutation	
permprogram(programname)	generate permutations of *permvar* by calling user-written *programname*
permprogramoptions(string)	pass *string* as options to *programname*
Reporting	
level(#)	set confidence level, default is level(95)
nodots	suppress replication dots
verbose	display full table legend
noisily	display any output from *command*
kdensityplot	plot the densities of each statistic in *exp_list*
noanalytics	do not send anonymized usage statistics to google analytics

weights are not allowed in *command* (they might work, but how they affect the results is unclear)

Description

ritest estimates p-values for permutation tests on the basis of Monte Carlo simulations. Unlike permute, ritest allows to specify more complex permutation structures, as generated for example by clustered treatment assignments.

Automatic permutation

Typing

```
ritest permvar exp_list, reps(#) strata(stratavar) cluster(clustervar): command
```

randomly permutes the values in permvar # times, respecting strata and clusters, each time executing command and collecting the associated values from the expressions in exp_list. Not specifying strata assumes no strata were used, which is equivalent to all observations being in one single stratum. Not

specifying clusters assumes no clusters were used, which is equivalent to each observations being a separate cluster.

File-based permutation

Typing

```
ritest permvar exp_list, reps(1000) permfile(permutations.dta) permmatchvar(id): command
```

merges the data based on id using permutations.dta (1:1, or if ids are not unique m:1). It then executes command each time, replacing permvar with permvar1, permvar2, permvar3, …, permvar1000, which have to be defined in permutations.dta prior to executing ritest.

These p-value estimates can be one-sided: Pr(T* < T) or Pr(T* > T). The default is two-sided: Pr(|T*| > |T|). Here T* denotes the value of the statistic from a randomly permuted dataset, and T denotes the statistic as computed on the original data.

permvar identifies the variable whose observed values will be randomly permuted.

command defines the statistical command to be executed. Most Stata commands and user-written programs can be used with ritest, as long as they follow standard Stata syntax. The by prefix may not be part of command.

exp_list specifies the statistics to be collected from the execution of command.

Program-based permutation

Typing

```
ritest t _b[t], nodots permprogram(permme) permprogramoptions("stratvar(strata)
clustvar(cluster)") r(`RR') : reg y x t
```

Reassigns *permvar* by calling a user-written program permme, and passing the options ("stratvar(strata) clustvar(cluster)"). You can always use a user-written program that permutes the relevant variable (e.g. mimicking the original treatment assignment procedure). The program will to be passed at least two parameters as options: *permvar* is the variable name of the relevant variable and *run* is an integer containing the iteration. *ritest* can be requested to also pass other options to the program using *permprogramoptions*.

Note: Both, automatic and file-based permutation, simply wrap program-based permutation with pre-specified programs.

Other options

kdensityplot produces a density plot of the realizations of expressions in *exp_list.* The realization for the expression in the original data is drawn as a vertical line.

noanalytics suppresses the sending of anonymous usage statistics. If this is not specified and the computer is connected to the internet, ritest will send a beacon to the author's google analytics account, containing information on (i) the version of stata, (ii) the version of ritest, (iii) the operating system, and (iv) whether program-based, file-based, or automatic permutation was used. No information on the command used, the data used, or any other information regarding the user or the data analysis is recorded. If you want to confirm that no other information is being sent, search the .ado-file for "GOOGLE-ANALYTICS" to find the part of the code related to this.

Code examples

Assume the data consists of observations from 2 schools, each school has 4 classes of 10 students each. Classes were randomly assigned to treatment, stratifying by school. This assured that each school had two treatment and to control classes. The outcome are test scores and the researcher wants to do a fisher test for the sharp null of no treatment effect, in a regression controlling for gender.

Automatic permutation

```
ritest treatment_b[treatment], cluster(classid) strata(schoolid): reg testscore treatment age
```

File-based permutation

Alternatively, an external file with permutations of treatment could be generated and used

```
# assuming you start with an empty datasets of classes containing
# only schoolid and classid
forvalues i = 1/100 {
                tempvar random cutoff
                gen `random'=rnormal()
                bys schoolid: egen `cutoff' = median(`random')
                gen treatment`i' = `random'> `cutoff'
        }
save permutations.dta
ritest treatment _b[treatment], permfile(permutations.dta) permmatchvar(classid)
```

The benefit of the file-based approach is that more complicated randomization structures can be implemented. For smaller samples, it is also straightforward to implement complete (non-stochastic) Fisher tests, which exhaust all possible permutations.

Program-based permutation

A user-written program can also be used to re-randomize:

```
program permme                          ///define the program
        syntax ,                        ///
                permvar(varname)        ///<- name of the permutation variable
                stratvar(varname)       ///<- name of the strata variable
                clustvar(varname)       ///<- name of the cluster variable
                *                       ///<- ritest also passes other things to the
                                        ///     permutation procedure (e.g. run(#))

        tempvar rr mr r
        // draw one random variable per cluster
        qui bys `clustvar': gen `r'=rnormal() if _n==1
        // compute median of these random variables within cluster
        qui bys `stratvar': egen `mr' = median(`r') if !missing(`r')
        //  impute random variable for all observations within clusters
        qui bys `clustvar': egen `rr' = mode(`r')
        // replace the permutation var with the new randomization outcome
        replace `permvar' = `rr'>`mr'
end
//no call ritest
ritest t _b[t], permprogram(permme) ///
                permprogramoptions("stratvar(schoolid) clustvar(classid)") ///
                : reg y x t
```

This method can be used implement any kind of re-randomization.

Author

Simon Heß, Goethe University Frankfurt, (hess@econ.uni-frankfurt.de)

~~The latest version of ritest can always be obtained from http://HessS.org.~~

I am happy to receive comments and suggestions regarding bugs or possibilities for improvements/extensions.