Title

stata.com

permute — Monte Carlo permutation tests

Description

`permute` estimates p-values for permutation tests on the basis of Monte Carlo simulations. Typing

```
. permute permvar exp_list, reps(#): command
```

randomly permutes the values in *permvar* # times, each time executing *command* and collecting the associated values from the expression in *exp_list*.

These p-value estimates can be one-sided: $\Pr(T^* \leq T)$ or $\Pr(T^* \geq T)$. The default is two-sided: $\Pr(|T^*| \geq |T|)$. 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 `permute`, as long as they follow standard Stata syntax; see **[U] 11 Language syntax**. The `by` prefix may not be part of *command*.

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

`permute` may be used for replaying results, but this feature is appropriate only when a dataset generated by `permute` is currently in memory or is identified by the `using` option. The variables specified in *varlist* in this context must be present in the respective dataset.

Quick start

Estimate p-values for a permutation test of $\Pr(|T^*| \geq |T|)$ for the sum of `y` where `x` > 0 using `nodrop` so that permutation of `y` is performed using all observations

```
permute y sum=r(sum), nodrop: sum y if x
```

As above, but with a one-tailed test for $\Pr(T^* \geq T)$

```
permute y sum=r(sum), nodrop right: sum y if x
```

As above, but with a one-tailed test for $\Pr(T^* \leq T)$

```
permute y sum=r(sum), nodrop left: sum y if x
```

Test of `stat=r(mystat)` returned by program `myprog` permuting values of `y`

```
permute y stat=r(mystat): myprog
```

As above, but use 500 replications

```
permute y stat=r(mystat), reps(500): myprog
```

As above, and save the replicates for each statistic in `myfile.dta`

```
permute y stat=r(mystat), reps(500) saving(myfile): myprog
```

Test of `stat1=r(mystat1)` and `stat2=r(mystat2)` returned by `myprog`

```
permute y stat1=r(mystat1) stat2=r(mystat2): myprog
```

Perform permutations within strata defined by `svar`

```
permute y stat=r(mystat), strata(svar): myprog
```

Menu

Statistics > Resampling > Permutation tests

Syntax

Compute permutation test

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

Report saved results

```
permute [ varlist ] [ using filename ] [ , display_options ]
```

options	Description
Main	
`reps(#)`	perform # random permutations; default is `reps(100)`
`left` | `right`	compute one-sided p-values; default is two-sided
Options	
`strata(`*varlist*`)`	permute within strata
`saving(`*filename*`, ...)`	save results to *filename*; save statistics in double precision; save results to *filename* every # replications
Reporting	
`level(#)`	set confidence level; default is `level(95)`
`noheader`	suppress table header
`nolegend`	suppress table legend
`verbose`	display full table legend
`nodrop`	do not drop observations
`nodots`	suppress replication dots
`dots(#)`	display dots every # replications
`noisily`	display any output from *command*
`trace`	trace *command*
`title(`*text*`)`	use *text* as title for permutation results
Advanced	
`eps(#)`	numerical tolerance; seldom used
`nowarn`	do not warn when `e(sample)` is not set
`force`	do not check for *weights* or `svy` commands; seldom used
`reject(`*exp*`)`	identify invalid results
`seed(#)`	set random-number seed to #

weights are not allowed in *command*.

display_options	Description
`left` | `right`	compute one-sided p-values; default is two-sided
`level(#)`	set confidence level; default is `level(95)`
`noheader`	suppress table header
`nolegend`	suppress table legend
`verbose`	display full table legend
`title(`*text*`)`	use *text* as title for results
`eps(#)`	numerical tolerance; seldom used

exp_list contains	(*name*: *elist*)
	elist
	eexp
elist contains	*newvar* = (*exp*)
	(*exp*)
eexp is	*specname*
	[*eqno*]*specname*
specname is	_b
	_b[]
	_se
	_se[]
eqno is	##
	name

exp is a standard Stata expression; see **[U] 13 Functions and expressions**.

Distinguish between `[]`, which are to be typed, and $[\,]$, which indicate optional arguments.

Options

Main

`reps(#)` specifies the number of random permutations to perform. The default is 100.

`left` or `right` requests that one-sided p-values be computed. If `left` is specified, an estimate of $\Pr(T^* \leq T)$ is produced, where T^* is the test statistic and T is its observed value. If `right` is specified, an estimate of $\Pr(T^* \geq T)$ is produced. By default, two-sided p-values are computed; that is, $\Pr(|T^*| \geq |T|)$ is estimated.

Options

`strata(`*varlist*`)` specifies that the permutations be performed within each stratum defined by the values of *varlist*.

`saving(`*filename*$[$`,` *suboptions*$]$`)` creates a Stata data file (`.dta` file) consisting of (for each statistic in *exp_list*) a variable containing the replicates.

`double` specifies that the results for each replication be saved as `doubles`, meaning 8-byte reals. By default, they are saved as `floats`, meaning 4-byte reals.

every(#) specifies that results are to be written to disk every #th replication. every() should be specified only in conjunction with saving() when *command* takes a long time for each replication. This will allow recovery of partial results should some other software crash your computer. See [P] **postfile**.

replace specifies that *filename* be overwritten if it exists. This option does not appear in the dialog box.

Reporting

level(#) specifies the confidence level, as a percentage, for confidence intervals. The default is level(95) or as set by set level; see [R] **level**.

noheader suppresses display of the table header. This option implies the nolegend option.

nolegend suppresses display of the table legend. The table legend identifies the rows of the table with the expressions they represent.

verbose requests that the full table legend be displayed. By default, coefficients and standard errors are not displayed.

nodrop prevents permute from dropping observations outside the if and in qualifiers. nodrop will also cause permute to ignore the contents of e(sample) if it exists as a result of running *command*. By default, permute temporarily drops out-of-sample observations.

nodots suppresses display of the replication dots. By default, one dot character is displayed for each successful replication. A red 'x' is displayed if *command* returns an error or if one of the values in *exp_list* is missing.

dots(#) displays dots every # replications. dots(0) is a synonym for nodots.

noisily requests that any output from *command* be displayed. This option implies the nodots option.

trace causes a trace of the execution of *command* to be displayed. This option implies the noisily option.

title(*text*) specifies a title to be displayed above the table of permutation results; the default title is Monte Carlo permutation results.

Advanced

eps(#) specifies the numerical tolerance for testing $|T^*| \geq |T|$, $T^* \leq T$, or $T^* \geq T$. These are considered true if, respectively, $|T^*| \geq |T| - \#$, $T^* \leq T + \#$, or $T^* \geq T - \#$. The default is 1e-7. You will not have to specify eps() under normal circumstances.

nowarn suppresses the printing of a warning message when *command* does not set e(sample).

force suppresses the restriction that *command* may not specify weights or be a svy command. permute is not suited for weighted estimation, thus permute should not be used with weights or svy. permute reports an error when it encounters weights or svy in *command* if the force option is not specified. This is a seldom used option, so use it only if you know what you are doing!

reject(*exp*) identifies an expression that indicates when results should be rejected. When *exp* is true, the resulting values are reset to missing values.

seed(#) sets the random-number seed. Specifying this option is equivalent to typing the following command prior to calling permute:

```
. set seed #
```

Remarks and examples

stata.com

Permutation tests determine the significance of the observed value of a test statistic in light of rearranging the order (permuting) of the observed values of a variable.

▷ Example 1: A simple two-sample test

Suppose that we conducted an experiment to determine the effect of a treatment on the development of cells. Further suppose that we are restricted to six experimental units because of the extreme cost of the experiment. Thus three units are to be given a placebo, and three units are given the treatment. The measurement is the number of newly developed healthy cells. The following listing gives the hypothetical data, along with some summary statistics.

```
. input y treatment
            y  treatment
  1. 7 0
  2. 9 0
  3. 11 0
  4. 10 1
  5. 12 1
  6. 14 1
  7. end
. sort treatment
. summarize y
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
           y |          6        10.5    2.428992         7         14
. by treatment: summarize y
-------------------------------------------------------------------------------
-> treatment = 0
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
           y |          3           9           2         7         11
-------------------------------------------------------------------------------
-> treatment = 1
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
           y |          3          12           2        10         14
```

Clearly, there are more cells in the treatment group than in the placebo group, but a statistical test is needed to conclude that the treatment does affect the development of cells. If the sum of the treatment measures is our test statistic, we can use `permute` to determine the probability of observing 36 or more cells, given the observed data and assuming that there is no effect due to the treatment.

```
. set seed 1234
. permute y sum=r(sum), saving(permdish) right nodrop nowarn: sum y if treatment
(running summarize on estimation sample)
Permutation replications (100)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
..................................................    50
..................................................   100
Monte Carlo permutation results                 Number of obs     =          6

      command:  summarize y if treatment
          sum:  r(sum)
  permute var:  y
```

T	T(obs)	c	n	p=c/n	SE(p)	[95% Conf.	Interval]
sum	36	6	100	0.0600	0.0237	.0223349	.1260299

```
Note: Confidence interval is with respect to p=c/n.
Note: c = #{T >= T(obs)}
```

We see that 6 of the 100 randomly permuted datasets yielded sums from the treatment group larger than or equal to the observed sum of 36. Thus the evidence is not strong enough, at the 5% level, to reject the null hypothesis that there is no effect of the treatment.

Because of the small size of this experiment, we could have calculated the exact permutation p-value from all possible permutations. There are six units, but we want the sum of the treatment units. Thus there are $\binom{6}{3} = 20$ permutation sums from the possible unique permutations.

$$
\begin{array}{cccc}
7+9+10=26 & 7+10+12=29 & 9+10+11=30 & 9+12+14=35 \\
7+9+11=27 & 7+10+14=31 & 9+10+12=31 & 10+11+12=33 \\
7+9+12=28 & 7+11+12=30 & 9+10+14=33 & 10+11+14=35 \\
7+9+14=30 & 7+11+14=32 & 9+11+12=32 & 10+12+14=36 \\
7+10+11=28 & 7+12+14=33 & 9+11+14=34 & 11+12+14=37
\end{array}
$$

Two of the 20 permutation sums are greater than or equal to 36. Thus the exact p-value for this permutation test is 0.1. Tied values will decrease the number of unique permutations.

When the `saving()` option is supplied, `permute` saves the values of the permutation statistic to the indicated file, in our case, `permdish.dta`. This file can be used to replay the result of `permute`. The `level()` option controls the confidence level of the confidence interval for the permutation p-value. This confidence interval is calculated using `cii` with the reported `n` (number of nonmissing replications) and `c` (the counter for events of significance).

```
. permute using permdish, level(80)
Monte Carlo permutation results                 Number of obs     =          6

      command:  summarize y if treatment
          sum:  r(sum)
  permute var:  y
```

T	T(obs)	c	n	p=c/n	SE(p)	[80% Conf.	Interval]
sum	36	6	100	0.0600	0.0237	.0318172	.1029391

```
Note: Confidence interval is with respect to p=c/n.
Note: c = #{|T| >= |T(obs)|}
```

◁

Example 2: Permutation tests with ANOVA

Consider some fictional data from a randomized complete-block design in which we wish to determine the significance of five treatments.

```
. use http://www.stata-press.com/data/r15/permute1, clear
. list y treatment in 1/10, abbrev(10)
```

	y	treatment
1.	4.407557	1
2.	5.693386	1
3.	7.099699	1
4.	3.12132	1
5.	5.242648	1
6.	4.280349	2
7.	4.508785	2
8.	4.079967	2
9.	5.904368	2
10.	3.010556	2

These data may be analyzed using `anova`.

```
. anova y treatment subject
```

```
                   Number of obs =         50    R-squared     =  0.3544
                   Root MSE      =    .914159    Adj R-squared =  0.1213
```

Source	Partial SS	df	MS	F	Prob>F
Model	16.518219	13	1.2706322	1.52	0.1574
treatment	13.022671	9	1.4469634	1.73	0.1174
subject	3.4955481	4	.87388703	1.05	0.3973
Residual	30.08475	36	.83568751		
Total	46.602969	49	.951081		

Suppose that we want to compute the significance of the F statistic for `treatment` by using `permute`. All we need to do is write a short program that will save the result of this statistic for `permute` to use. For example,

```
program panova, rclass
        version 15.1
        args response fac_intrst fac_other
        anova `response' `fac_intrst' `fac_other'
        return scalar Fmodel = e(F)
        test `fac_intrst'
        return scalar F = r(F)
end
```

Now in `panova`, `test` saves the F statistic for the factor of interest in `r(F)`. This is different from `e(F)`, which is the overall model F statistic for the model fit by `anova` that `panova` saves in `r(Fmodel)`. In the following example, we use the `strata()` option so that the treatments are randomly rearranged within each `subject`. It should not be too surprising that the estimated p-values are equal for this example, because the two F statistics are equivalent when controlling for differences between subjects. However, we would not expect to always get the same p-values every time we reran `permute`.

```
. set seed 1234
. permute treatment treatmentF=r(F) modelF=e(F), reps(1000) strata(subject)
> saving(permanova) nodots: panova y treatment subject
Monte Carlo permutation results
Number of strata =          5                  Number of obs     =         50
      command:  panova y treatment subject
   treatmentF:  r(F)
       modelF:  e(F)
  permute var:  treatment
```

T	T(obs)	c	n	p=c/n	SE(p)	[95% Conf.	Interval]
treatmentF	1.731465	133	1000	0.1330	0.0107	.112558	.1556293
modelF	1.520463	133	1000	0.1330	0.0107	.112558	.1556293

```
Note: Confidence intervals are with respect to p=c/n.
Note: c = #{|T| >= |T(obs)|}
```

◁

▷ Example 3: Wilcoxon rank-sum test

As a final example, let's consider estimating the p-value of the Z statistic returned by `ranksum`. Suppose that we collected data from some experiment: `y` is some measure we took on 17 individuals, and `group` identifies the group that an individual belongs to.

```
. use http://www.stata-press.com/data/r15/permute2
. list
```

	group	y
1.	1	6
2.	1	11
3.	1	20
4.	1	2
5.	1	9
6.	1	5
7.	0	2
8.	0	1
9.	0	6
10.	0	0
11.	0	2
12.	0	3
13.	0	3
14.	0	12
15.	0	4
16.	0	1
17.	0	5

Next we analyze the data using `ranksum` and notice that the observed value of the test statistic (stored as `r(z)`) is -2.02 with an approximate p-value of 0.0434.

```
. ranksum y, by(group)
Two-sample Wilcoxon rank-sum (Mann-Whitney) test
       group |       obs    rank sum    expected
-------------+---------------------------------
           0 |        11          79          99
           1 |         6          74          54
-------------+---------------------------------
    combined |        17         153         153
unadjusted variance       99.00
adjustment for ties       -0.97
                          ------
adjusted variance         98.03
Ho: y(group==0) = y(group==1)
             z =  -2.020
    Prob > |z| =   0.0434
```

The observed value of the rank-sum statistic is 79, with an expected value (under the null hypothesis of no group effect) of 99. There are 17 observations, so the permutation distribution contains $\binom{17}{6} = 12{,}376$ possible values of the rank-sum statistic if we ignore ties. With ties, we have fewer possible values but still too many to want to count them. Thus we use `permute` with 10,000 replications and see that the Monte Carlo permutation test agrees with the result of the test based on the normal approximation.

```
. set seed 18385766
. permute y z=r(z), reps(10000) nowarn nodots: ranksum y, by(group)
Monte Carlo permutation results                 Number of obs     =         17
      command:  ranksum y, by(group)
            z:  r(z)
  permute var:  y
------------------------------------------------------------------------------
T             |     T(obs)       c       n   p=c/n   SE(p) [95% Conf. Interval]
--------------+---------------------------------------------------------------
          z   | -2.020002     455   10000  0.0455  0.0021   .0414984   .0497689
------------------------------------------------------------------------------
Note: Confidence interval is with respect to p=c/n.
Note: c = #{|T| >= |T(obs)|}
```

◁

For an application of a permutation test to a problem in epidemiology, see Hayes and Moulton (2017, 237–241).

❑ Technical note

`permute` reports confidence intervals for p to emphasize that it is based on the binomial estimator for proportions. When the variability implied by the confidence interval makes conclusions difficult, you may increase the number of replications to determine more precisely the significance of the test statistic of interest. In other words, the value of p from `permute` will converge to the true permutation p-value as the number of replications gets arbitrarily large. ❑

Stored results

permute stores the following in r():

Scalars

r(N)	sample size	r(k_exp)	number of standard expressions
r(N_reps)	number of requested replications	r(k_eexp)	number of _b/_se expressions
r(level)	confidence level		

Macros

r(cmd)	permute	r(left)	left or empty
r(command)	*command* following colon	r(right)	right or empty
r(permvar)	permutation variable	r(rngstate)	random-number state used
r(title)	title in output	r(event)	T <= T(obs), T >= T(obs), or |T| <= |T(obs)|
r(exp#)	#th expression		

Matrices

r(b)	observed statistics	r(p)	observed proportions
r(c)	count when r(event) is true	r(se)	standard errors of observed proportions
r(reps)	number of nonmissing results	r(ci)	confidence intervals of observed proportions

References

Ängquist, L. 2010. Stata tip 92: Manual implementation of permutations and bootstraps. *Stata Journal* 10: 686–688.

Gallis, J. A., F. Li, H. Yu, and E. L. Turner. 2018. cvcrand and cptest: Commands for efficient design and analysis of cluster randomized trials using constrained randomization and permutation tests. *Stata Journal* 18: 357–378.

Good, P. I. 2006. *Resampling Methods: A Practical Guide to Data Analysis*. 3rd ed. Boston: Birkhäuser.

Hayes, R. J., and L. H. Moulton. 2017. *Cluster Randomised Trials*. 2nd ed. Boca Raton, FL: CRC Press.

Kaiser, J. 2007. An exact and a Monte Carlo proposal to the Fisher–Pitman permutation tests for paired replicates and for independent samples. *Stata Journal* 7: 402–412.

Kaiser, J., and M. G. Lacy. 2009. A general-purpose method for two-group randomization tests. *Stata Journal* 9: 70–85.

Also see

[R] **bootstrap** — Bootstrap sampling and estimation

[R] **jackknife** — Jackknife estimation

[R] **simulate** — Monte Carlo simulations