Description

`permute` estimates *p*-values for permutation tests using 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 expressions in `exp_list`.

`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 specification.

Quick start

Estimate *p*-values for a permutation test of the coefficient of *x* in a linear regression, permuting values of the outcome *y*

```
permute y _b[x]: regress y x
```

Test for `r(mystat)` returned by program `myprog`, permuting values of *y*

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

As above, but increase the number of permutations from the default of 100 to 500

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

As above, but set the random-number seed for reproducibility, and save the permuted statistics in `myfile.dta`

```
permute y r(mystat), reps(500) seed(1234) saving(myfile): myprog
```

Test for `r(mystat1)` and `r(mystat2)`, naming the statistics `stat1` and `stat2`, respectively

```
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

Perform permutation test

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

Report saved results

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

### options Description

<table>
<thead>
<tr>
<th>Main</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>reps(#)</code></td>
<td>perform # Monte Carlo permutations; default is <code>reps(100)</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Options</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>seed(#)</code></td>
<td>set random-number seed to #</td>
</tr>
<tr>
<td><code>strata(varlist)</code></td>
<td>permute within strata</td>
</tr>
<tr>
<td><code>saving(filename, ...)</code></td>
<td>save results to <code>filename</code>; save statistics in double precision; save results to <code>filename</code> every # permutations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reporting</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>level(#)</code></td>
<td>set confidence level; default is <code>level(95)</code></td>
</tr>
<tr>
<td><code>noheader</code></td>
<td>suppress table header</td>
</tr>
<tr>
<td><code>nolegend</code></td>
<td>suppress table legend</td>
</tr>
<tr>
<td><code>verbose</code></td>
<td>display full table legend</td>
</tr>
<tr>
<td><code>title(text)</code></td>
<td>use <code>text</code> as title for permutation results</td>
</tr>
<tr>
<td><code>dots(#)</code></td>
<td>display dots every # permutations</td>
</tr>
<tr>
<td><code>nodots</code></td>
<td>suppress permutation dots</td>
</tr>
<tr>
<td><code>noisily</code></td>
<td>display any output from <code>command</code></td>
</tr>
<tr>
<td><code>trace</code></td>
<td>trace <code>command</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Advanced</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nodrop</code></td>
<td>do not drop observations</td>
</tr>
<tr>
<td><code>nowarn</code></td>
<td>do not warn when <code>e(sample)</code> is not set</td>
</tr>
<tr>
<td><code>force</code></td>
<td>do not check for <code>weights</code> or <code>svy</code> commands; seldom used</td>
</tr>
<tr>
<td><code>reject(exp)</code></td>
<td>identify invalid results</td>
</tr>
<tr>
<td><code>eps(#)</code></td>
<td>numerical tolerance; seldom used</td>
</tr>
</tbody>
</table>

`weights` are not allowed in `command` unless the `force` option is specified.
**display_options**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>level(#)</strong></td>
</tr>
<tr>
<td><strong>noheader</strong></td>
</tr>
<tr>
<td><strong>nolegend</strong></td>
</tr>
<tr>
<td><strong>verbose</strong></td>
</tr>
<tr>
<td><strong>title(text)</strong></td>
</tr>
<tr>
<td><strong>eps(#)</strong></td>
</tr>
</tbody>
</table>

**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 Monte Carlo permutations to perform. The default is `reps(100)`.

The default of 100 permutations is chosen for convenience. In real-world applications, you will most likely need to use more permutations. `permute` reports Monte Carlo error that you can use to evaluate whether the specified number of permutations provides sufficient precision for the reported p-value estimates.

**Options**

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

  ```stata
  . set seed #
  ```

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

- `saving(filename[, double every(#) replace])` creates a Stata data file (.dta file) consisting of variables for each statistic in `exp_list` containing the results for each permutation.
double specifies that the results for each permutation be saved as doubles, meaning 8-byte reals. By default, they are saved as floats, meaning 4-byte reals.

every(#) specifies that results be written to disk every #th permutation. every() should be specified only in conjunction with saving() when command takes a long time for each permutation. 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.

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.

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

dots(#) and nodots specify whether to display permutation dots. By default, one dot character is displayed for each successful permutation. A red ‘x’ is displayed if command returns an error or if any value in exp_list is missing. You can also control whether dots are displayed using set dots; see [R] set.

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

nodots suppresses display of the permutation dots.

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.

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.

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 an 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.

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

eps(#) specifies the numerical tolerance for testing $T \leq T_{obs}$ and $T \geq T_{obs}$, where $T$ is the test statistic and $T_{obs}$ is its observed value. These are considered true if, respectively, $T \leq T_{obs} + #$ or $T \geq T_{obs} - #$. The default is eps(1e-7). You will not have to specify eps() under normal circumstances.
Remarks and examples

Remarks are presented under the following headings:

Introduction
A first example
Two-sided p-values from permutation tests
One-sided permutation test

Introduction

Permutation tests are based on the idea of scrambling—that is, permuting—the order of a variable in all possible ways, calculating the value of a test statistic for each permutation, and taking this set of values of the statistic as its distribution.

For instance, consider the correlation of two variables, \( \text{corr}(x, y) \), where \( x = (x_1, \ldots, x_n) \) and \( y = (y_1, \ldots, y_n) \). We hold the order of \( x \) fixed and permute the order of \( y \) in all possible ways. For each permutation \( y^* \), we calculate \( T^* = \text{corr}(x, y^*) \). The set of \( T^* \) gives the permutation distribution for the correlation. This permutation distribution is the distribution of the correlation under the null hypothesis that the ordering of the elements of \( y \) are independent of the ordering of the elements of \( x \), conditional on the observed values of \( x \) and \( y \).

Aside: Actually, the null hypothesis does not require independence. A weaker assumption of exchangeability is sufficient. If \( x \) and \( y \) are observed values of the random variates \( X = (X_1, \ldots, X_n) \) and \( Y = (Y_1, \ldots, Y_n) \), then the joint distribution \( f(X, Y) \) is called exchangeable when it is invariant to the orderings of \( X_1, \ldots, X_n \) and \( Y_1, \ldots, Y_n \).

The \( p \)-value for the permutation test is the proportion of permutations that produces a test statistic \( T^* \) as extreme or more extreme than the test statistic \( T_{obs} \) computed using the observed data.

A first example

We first demonstrate how to apply the `permute` prefix by testing for a difference in the distribution of a variable across two groups.

Example 1: Wilcoxon rank-sum test

Let’s consider calculating the \( p \)-value for the Wilcoxon rank-sum test performed by `ranksum`. Suppose that we collected data from some experiment: \( y \) is some measure we took on 17 individuals, and \( group \) identifies the group to which an individual belongs.
We analyze the data using `ranksum`:

```
. ranksum y, by(group)
```

Two-sample Wilcoxon rank-sum (Mann-Whitney) test

<table>
<thead>
<tr>
<th>group</th>
<th>obs</th>
<th>rank sum</th>
<th>expected</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11</td>
<td>79</td>
<td>99</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>74</td>
<td>54</td>
</tr>
<tr>
<td>combined</td>
<td>17</td>
<td>153</td>
<td>153</td>
</tr>
</tbody>
</table>

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
Exact Prob = 0.0436

The test gives an approximate \( p \)-value of 0.0434 and an exact \( p \)-value of 0.0436.

Let's try to reproduce these results using `permute`. The test statistic \( T \) for the Wilcoxon rank-sum test is the sum of the ranks for the first group, which is 79, and is stored as \( r(sum\_obs) \). We specify `reps(10000)` to do 10,000 Monte Carlo permutations and `dots(100)` to display a dot every 100th permutation. We set the random-number seed so we can duplicate our results.
. set seed 1234
. permute group r(sum_obs), reps(10000) dots(100): ranksum y, by(group)
(running ranksum on estimation sample)

Warning: Because ranksum is not an estimation command or does not set e(sample), permute has no way to determine which observations are used in calculating the statistics and so assumes that all observations are used. This means that no observations will be excluded from the resampling because of missing values or other reasons.

If the assumption is not true, press Break, save the data, and drop the observations that are to be excluded. Be sure that the dataset in memory contains only the relevant data.

Monte Carlo permutation results

<table>
<thead>
<tr>
<th></th>
<th>Monte Carlo error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T(obs) Test c n p SE(p) [95% CI(p)]</td>
</tr>
<tr>
<td>pm_1</td>
<td>79 lower 230 10000 .0230 .0015 .0202 .0261</td>
</tr>
<tr>
<td></td>
<td>upper 9796 10000 .9796 .0014 .9766 .9823</td>
</tr>
<tr>
<td></td>
<td>two-sided .0460 .0021 .0419 .0501</td>
</tr>
</tbody>
</table>

Note: For lower one-sided test, c = #{T <= T(obs)} and p = p_lower = c/n.
Note: For upper one-sided test, c = #{T >= T(obs)} and p = p_upper = c/n.
Note: For two-sided test, p = 2*min(p_lower, p_upper); SE and CI approximate.

The lengthy message about e(sample) is worth noting. If there were missing values in the data, we might want to drop those observations before running permute. To suppress the message in future runs, use the nowarn option.

permute displays three different p-values: the lower and upper one-sided p-values and the two-sided p-value. Here, the two-sided p-value obtained by this Monte Carlo procedure is 0.0460, which is close to the exact p-value of 0.0436 computed by ranksum. See the next section for a description of how two-sided p-values are calculated when performing permutation tests. See example 2 for a test that requires a one-sided p-value.

permute reports standard errors and confidence intervals for p-values because, as with any other Monte Carlo procedure, they are approximations to the true exact p-values. These statistics are useful to assess the precision of the computed p-values. If you need more precision, specify more permutations in the reps() option. See Methods and formulas for a description of how the standard errors and confidence intervals are calculated.

The confidence interval for the Monte Carlo two-sided p-value in this example is [0.0419, 0.0501]. If we did not have recourse to the exact value, we would likely want to run permute again with an even higher number of random permutations to narrow the confidence interval.
Two-sided p-values from permutation tests

In the above example, we used the two-sided $p$-value for our hypothesis testing. For permutation distributions, two-sided $p$-values require some explanation about how they are calculated. permute calculates the two-sided $p$-value as $p = 2 \min(p_{\text{lower}}, p_{\text{upper}})$, where $p_{\text{lower}}$ is the lower one-sided $p$-value and $p_{\text{upper}}$ the upper one-sided $p$-value. (More precisely, $p = \min[1, 2 \min(p_{\text{lower}}, p_{\text{upper}})]$ is used because obviously $p$-values must be bounded by 1.)

In general, the $p$-value is defined as the probability under the null hypothesis of obtaining a value of the test statistic $T$ equal to or more extreme than the value $T_{\text{obs}}$ that was actually observed. For one-sided $p$-values, what is “more extreme” is clear. For lower one-sided $p$-values, it is the probability that $T \leq T_{\text{obs}}$, and for upper one-sided $p$-values, it is the probability that $T \geq T_{\text{obs}}$. When $T$ has a symmetric distribution, the two-sided $p$-value is typically defined as the probability that $|T| \geq |T_{\text{obs}}|$. Permutation distributions, however, are not in general symmetric.

Under a permutation-based null hypothesis, the domain of $T$ consists of all the possible permutations of the underlying data used to calculate $T$. The domain is discrete and finite, and hence the permutation distribution of $T$ is discrete and finite. These finite distributions are symmetric only in certain cases. For instance, with our example of the Wilcoxon rank-sum test, if the data consist of untied ranks and $T$ is the sum of a subsample of the ranks, the distribution is symmetric. When there are ties in the ranks, however, the distribution is in most cases not symmetric.

When distributions are asymmetric, what values of $T$ are “more extreme” than $T_{\text{obs}}$? Suppose $T_{\text{obs}}$ is below the mean of the distribution. Clearly, the lower-tail values $T \leq T_{\text{obs}}$ are more extreme. But what values of $T$ from the upper tail are more extreme?

For asymmetric distributions, the rationale for using $p = 2 \min(p_{\text{lower}}, p_{\text{upper}})$ for two-sided tests is the following: It takes the smallest one-sided $p$-value and doubles it. Comparing this two-sided $p$-value against a significance level of, say, 0.05 is equivalent to comparing the smallest one-sided $p$-value against a level of 0.025. It essentially turns the two-sided test into a one-sided test with the significance level cut in half. So this definition conveniently sidesteps the need to define what values of $T$ from the opposite tail from $T_{\text{obs}}$ are more extreme! Also, it is appropriate for both symmetric and asymmetric distributions.

One-sided permutation test

In some cases, we perform a permutation test based on a one-sided $p$-value.

Example 2: Permutation tests with ANOVA

Consider some fictional data from an experimental randomized complete-block design in which there are 5 subjects each receiving 10 different treatments. We want to test whether any of the treatments have an effect different from the effects of the other treatments.
Let's load the data and list the data for the first two subjects:

```
. use https://www.stata-press.com/data/r16/permute1, clear
. sort subject treatment
. list subject treatment y in 1/20, abbrev(10)
```

<table>
<thead>
<tr>
<th>subject</th>
<th>treatment</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4.407557</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4.280349</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4.418574</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4.075359</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>3.899775</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>5.533271</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>5.142111</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>5.791124</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>4.504411</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>4.896333</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>5.693386</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>4.508785</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>5.10376</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>5.753985</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>5.092277</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>4.496496</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>6.339948</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>4.820389</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>5.686253</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>6.951727</td>
</tr>
</tbody>
</table>

These data may be analyzed using `anova`.

```
. anova y treatment subject
```

```
<table>
<thead>
<tr>
<th>Source</th>
<th>Partial SS</th>
<th>df</th>
<th>MS</th>
<th>F</th>
<th>Prob&gt;F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>16.518219</td>
<td>13</td>
<td>1.2706322</td>
<td>1.52</td>
<td>0.1574</td>
</tr>
<tr>
<td>treatment</td>
<td>13.022671</td>
<td>9</td>
<td>1.4469634</td>
<td>1.73</td>
<td>0.1174</td>
</tr>
<tr>
<td>subject</td>
<td>3.4955481</td>
<td>4</td>
<td>0.87388703</td>
<td>1.05</td>
<td>0.3973</td>
</tr>
<tr>
<td>Residual</td>
<td>30.08475</td>
<td>36</td>
<td>.83568751</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>46.602969</td>
<td>49</td>
<td></td>
<td></td>
<td>.951081</td>
</tr>
</tbody>
</table>
```

Number of obs = 50
R-squared = 0.3544
Root MSE = .914159
Adj R-squared = 0.1213
anova gives a \( p \)-value of 0.1174 for the treatment effect. This \( p \)-value is calculated with the assumption of normality for the distribution of the outcome conditional on the means of each treatment and subject effect.

Suppose we do not want to assume normality. The treatments were assigned in a random order to each of the subjects. A null hypothesis of no treatment effect means that the observed values of \( y \) and their order were determined by factors other than the treatments. The treatments were essentially labels that had nothing to do with the outcomes, and any other ordering of the labels would be a possible occurrence. That is, we imagine running the experiment multiple times, each with a different ordering of the treatments, but each time, we get the same observed values of \( y \). This is the permutation-based formulation of the null hypothesis.

What about the subjects? Each subject gets each of the 10 treatments, so clearly we must permute the treatments within each subject independently of the permutations for the other subjects. We can do this using the strata() option with permute.

If we type ereturn list after anova, we see that the \( F \) statistic for treatment is stored in e(F_1). This is our test statistic for our permutation test.

We save the dataset containing all the permutations of the test statistic using the saving() option. Specifying the test statistic as \( F_{\text{treatment}} = e(F_1) \) labels the test statistic as \( F_{\text{treatment}} \) in the output and is also the name of the variable containing the test statistic in permanova.dta, the dataset created by saving(). We also specify the nodots option to suppress the dots in the output.

```
. set seed 1234
. permute treatment F_{treatment}=e(F_1), reps(10000) strata(subject) saving(permanova) nodots: anova y treatment subject
```

Monte Carlo permutation results

<table>
<thead>
<tr>
<th>Number of observations = 50</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of strata = 5</td>
<td></td>
</tr>
<tr>
<td>Number of permutations = 10,000</td>
<td></td>
</tr>
</tbody>
</table>

command: anova y treatment subject
F_{treatment}: e(F_1)
permute var: treatment

<table>
<thead>
<tr>
<th>T</th>
<th>T(obs)</th>
<th>Test</th>
<th>c</th>
<th>n</th>
<th>p</th>
<th>SE(p)</th>
<th>[95% CI(p)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_{treatment}</td>
<td>1.731465</td>
<td>lower</td>
<td>8814</td>
<td>10000</td>
<td>.8814</td>
<td>.0032</td>
<td>.8749 .8877</td>
</tr>
<tr>
<td></td>
<td></td>
<td>upper</td>
<td>1186</td>
<td>10000</td>
<td>.1186</td>
<td>.0032</td>
<td>.1123 .1251</td>
</tr>
<tr>
<td></td>
<td></td>
<td>two-sided</td>
<td></td>
<td></td>
<td>.2372</td>
<td>.0043</td>
<td>.2289 .2455</td>
</tr>
</tbody>
</table>

Note: For lower one-sided test, \( c = \#\{T \leq T(\text{obs})\} \) and \( p = p_{\text{lower}} = c/n \).
Note: For upper one-sided test, \( c = \#\{T \geq T(\text{obs})\} \) and \( p = p_{\text{upper}} = c/n \).
Note: For two-sided test, \( p = 2\min(p_{\text{lower}}, p_{\text{upper}}) \); SE and CI approximate.

Our test statistic is an \( F \) statistic, so we are interested in the number of permutations that have a larger (more extreme) statistic than the 1.73 we obtained with our original data. Therefore, we want the upper one-sided \( p \)-value, which is 0.1186. This value is close to the \( p \)-value given by anova of 0.1174 for the treatment effect.

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

permute stores the following in r():

Scalars
- r(N) number of observations for command
- r(n_reps) number of requested permutations in reps()
- r(k_exp) number of standard expressions
- r(k_eexp) number of _b and _se expressions
- r(n_strata) number of strata, if strata() specified
- r(level) confidence level

Macros
- r(cmd) permute
- r(command) command following colon
- r(permvar) permutation variable
- r(title) title in output
- r(rngstate) random-number state used
- r(exp#) #th expression
- r(strata) strata variable, if strata() specified
- r(missing) "missing" when one or more expressions equal missing value

Matrices
- r(b) observed statistics
- r(n) number of nonmissing results
- r(c_lower) counts for lower one-sided p-values
- r(c_upper) counts for upper one-sided p-values
- r(p_lower) lower one-sided p-values
- r(p_upper) upper one-sided p-values
- r(p_twosided) two-sided p-values
- r(se_p_lower) standard errors of lower one-sided p-values
- r(se_p_upper) standard errors of upper one-sided p-values
- r(se_p_twosided) standard errors of two-sided p-values
- r(ci_p_lower) confidence intervals of lower one-sided p-values
- r(ci_p_upper) confidence intervals of upper one-sided p-values
- r(ci_p_twosided) confidence intervals of two-sided p-values

Methods and formulas

One-sided p-values are based on counts of the test statistic $T$ calculated for each permutation that are more extreme than the observed value $T_{obs}$. The lower one-sided p-value uses the count $c = \#\{T \leq T_{obs}\}$, and the upper one-sided p-value uses $c = \#\{T \geq T_{obs}\}$.

These counts are assumed to have a binomial distribution. Standard errors and confidence intervals are computed using cii proportions $n \, c$, where $n$ is the number of permutations that yielded nonmissing results and $c$ is the count. The confidence intervals are exact binomial confidence intervals. See Methods and formulas in [R] ci.

permute calculates the two-sided p-value as $p = \min[1, 2 \min(p_{lower}, p_{upper})]$, where $p_{lower}$ is the lower one-sided p-value and $p_{upper}$ the upper one-sided p-value. Because this definition does not yield a simple formula for the standard error or confidence interval, the following ad hoc procedure is used. If $p_{lower}$ is the minimum one-sided p-value, its count $c_{lower}$ is doubled. If $p_{upper}$ is the minimum one-sided p-value, its count $c_{upper}$ is doubled. More precisely, the value $c_2 = \min[n, 2 \min(c_{lower}, c_{upper})]$ is used, and its distribution is assumed to be approximately binomial. Standard errors and confidence intervals are computed using cii proportions $n \, c_2$, wald. The confidence intervals produced are asymptotic binomial confidence intervals.
References


Also see

[R] bootstrap — Bootstrap sampling and estimation

[R] jackknife — Jackknife estimation

[R] simulate — Monte Carlo simulations