

cvpermute() — Obtain all permutations

[Description](#) [Syntax](#) [Remarks and examples](#) [Conformability](#)
[Diagnostics](#) [Also see](#)

Description

`cvpermute()` returns all permutations of the values of column vector V , one at a time. If $V = (1\ 2\ 3)$, there are six permutations and they are $(1\ 2\ 3)$, $(1\ 3\ 2)$, $(2\ 1\ 3)$, $(2\ 3\ 1)$, $(3\ 1\ 2)$, and $(3\ 2\ 1)$. If $V = (1\ 2\ 1)$, there are three permutations and they are $(1\ 1\ 2)$, $(1\ 2\ 1)$, and $(2\ 1\ 1)$.

Vector V is specified by calling `cvpermutesetup()`,

```
info = cvpermutesetup(V)
```

`info` holds information that is needed by `cvpermute()` and it is `info`, not V , that is passed to `cvpermute()`. To obtain the permutations, repeated calls are made to `cvpermute()` until it returns `J(0,1,.)`:

```
info = cvpermutesetup(V)
while ((p=cvpermute(info)) != J(0,1,.) ) {
    ... p ...
}
```

Column vector p will contain a permutation of V .

`cvpermutesetup()` may be specified with one or two arguments:

```
info = cvpermutesetup(V)
info = cvpermutesetup(V, unique)
```

`unique` is usually not specified. If `unique` is specified, it should be 0 or 1. Not specifying `unique` is equivalent to specifying `unique = 0`. Specifying `unique = 1` states that the elements of V are unique or, at least, are to be treated that way.

When the arguments of V are unique—for instance, $V = (1\ 2\ 3)$ —specifying `unique = 1` will make `cvpermute()` run faster. The same permutations will be returned, although usually in a different order.

When the arguments of V are not unique—for instance, $V = (1\ 2\ 1)$ —specifying `unique = 1` will make `cvpermute()` treat them as if they were unique. With `unique = 0`, there are three permutations of $(1\ 2\ 1)$. With `unique = 1`, there are six permutations, just as there are with $(1\ 2\ 3)$.

Syntax

```
info = cvpermutesetup(real colvector V [ , real scalar unique ])  
real colvector cvpermute(info)
```

where *info* should be declared *transmorphic*.

Remarks and examples

[stata.com](http://www.stata.com)

► Example 1

You have the following data:

v1	v2
22	29
17	33
21	26
20	32
16	35

You wish to do an exact permutation test for the correlation between *v1* and *v2*.

That is, you wish to (1) calculate the correlation between *v1* and *v2*—call that value *r*—and then (2) calculate the correlation between *v1* and *v2* for all permutations of *v1*, and count how many times the result is more extreme than *r*.

For the first step,

```
: X = (22, 29 \  
>      17, 33 \  
>      21, 26 \  
>      20, 32 \  
>      16, 35)  
  
:  
: correlation(X)  
[symmetric]
```

	1	2
1	1	
2	-.8468554653	1

The correlation is $-.846855$. For the second step,

```
: V1 = X[,1]  
: V2 = X[,2]  
: num = den = 0  
: info = cvpermutesetup(V1)  
: while ((V1=cvpermute(info)) != J(0,1,..)) {  
>     rho = correlation((V1,V2))[2,1]  
>     if (rho<=-.846 | rho>=.846) num++  
>     den++  
> }
```

```

: (num, den, num/den)
      1          2          3
1  ┌──────────┬──────────┬──────────┐
   │          13          120  .1083333333 │
   └──────────┴──────────┴──────────┘

```

Of the 120 permutations, 13 (10.8%) were outside .846855 or $-.846855$.



▶ Example 2

You now wish to do the same thing but using the Spearman rank-correlation coefficient. Mata has no function that will calculate that, but Stata has a command that does—see [R] [spearman](#)—so we will use the Stata command as our subroutine.

This time, we will assume that the data have been loaded into a Stata dataset:

```

. list

```

	var1	var2
1.	22	29
2.	17	33
3.	21	26
4.	20	32
5.	16	35

For the first step,

```

. spearman var1 var2
Number of obs =      5
Spearman's rho = -0.9000
Test of Ho: var1 and var2 are independent
Prob > |t| =      0.0374

```

For the second step,

```

. mata
----- mata (type end to exit) -----
: V1 = st_data(., "var1")
: info = cvpermutesetup(V1)
: num = den = 0
: while ((V1=cvpermute(info)) != J(0,1,.)) {
>   st_store(., "var1", V1)
>   stata("quietly spearman var1 var2")
>   rho = st_numscalar("r(rho)")
>   if (rho<=-.9 | rho>=.9) num++
>   den++
> }
: (num, den, num/den)
      1          2          3
1  ┌──────────┬──────────┬──────────┐
   │          2          120  .0166666667 │
   └──────────┴──────────┴──────────┘

```

Only two of the permutations resulted in a rank correlation of at least .9 in magnitude.

In the code above, we obtained the rank correlation from `r(rho)` which, we learned from [R] `spearman`, is where `spearman` stores it.

Also note how we replaced the contents of `var1` by using `st_store()`. Our code leaves the dataset changed and so could be improved.

◀

Conformability

`cvpermutesetup(V, unique)`:

V: $n \times 1$
unique: 1×1 (optional)
result: $1 \times L$

`cvpermute(info)`:

info: $1 \times L$
result: $n \times 1$ or 0×1

where

$$L = \begin{cases} 3 & \text{if } n = 0 \\ 4 & \text{if } n = 1 \\ (n + 3)(n + 2)/2 - 6 & \text{otherwise} \end{cases}$$

The value of L is not important except that the *info* vector returned by `cvpermutesetup()` and then passed to `cvpermute()` consumes memory. For instance,

<i>n</i>	<i>L</i>	Total memory ($8 * L$)
5	22	176 bytes
10	72	576
50	1,372	10,560
100	5,247	41,976
1,000	502,497	4,019,976

Diagnostics

`cvpermute()` returns `J(0,1,.)` when there are no more permutations.

Also see

[M-4] `statistical` — Statistical functions