cvpermute() — Obtain all permutations

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

\[ \text{info} = \text{cvpermutesetup}(\text{real colvector } V \ [\ , \ \text{real scalar unique}]) \]

\[ \text{real colvector cvpermute(info)} \]

where \( \text{info} \) should be declared transmorphic.

Remarks and examples

Example 1

You have the following data:

\[
\begin{array}{cc}
\hline 
v1 & v2 \\
22 & 29 \\
17 & 33 \\
21 & 26 \\
20 & 32 \\
16 & 35 \\
\end{array}
\]

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,

\[
: X = (22, 29 \ \backslash \\
> 17, 33 \ \backslash \\
> 21, 26 \ \backslash \\
> 20, 32 \ \backslash \\
> 16, 35) \\
:
: \text{correlation}(X) \\
[\text{symmetric}] \\
\begin{array}{cc}
1 & 2 \\
1 & 1 \\
2 & -.8468554653 & 1 \\
\end{array}
\]

The correlation is \(-.846855\). For the second step,

\[
: V1 = X[,1] \\
: V2 = X[,2] \\
: \text{num} = \text{den} = 0 \\
: \text{info} = \text{cvpermutesetup}(V1) \\
: \text{while} ((V1=\text{cvpermute(info)}) \neq J(0,1,\ldots)) \{ \\
> \quad \text{rho} = \text{correlation}((V1,V2))[2,1] \\
> \quad \text{if} (\text{rho}<=-.846 \ \text{\|} \ \text{rho}>.846) \quad \text{num}++ \\
> \quad \text{den}++ \\
> \} \
\]
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
+------------------+
<table>
<thead>
<tr>
<th>var1</th>
<th>var2</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>29</td>
</tr>
<tr>
<td>17</td>
<td>33</td>
</tr>
<tr>
<td>21</td>
<td>26</td>
</tr>
<tr>
<td>20</td>
<td>32</td>
</tr>
<tr>
<td>16</td>
<td>35</td>
</tr>
</tbody>
</table>
+------------------+
```

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)
+-------+-------+-------+
<table>
<thead>
<tr>
<th>num</th>
<th>den</th>
<th>num/den</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>120</td>
<td>0.0167</td>
</tr>
</tbody>
</table>
+-------+-------+---------+
```

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

\[
\text{cvpermutesetup}(V, unique):
\]
- \( V: n \times 1 \)
- \( unique: 1 \times 1 \) (optional)
- \( result: 1 \times L \)

\[
\text{cvpermute(info)}:
\]
- \( info: 1 \times L \)
- \( result: n \times 1 \) or \( 0 \times 1 \)

where

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

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

<table>
<thead>
<tr>
<th>( n )</th>
<th>( L )</th>
<th>Total memory ((8 \times L))</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>22</td>
<td>176 bytes</td>
</tr>
<tr>
<td>10</td>
<td>72</td>
<td>576</td>
</tr>
<tr>
<td>50</td>
<td>1,372</td>
<td>10,560</td>
</tr>
<tr>
<td>100</td>
<td>5,247</td>
<td>41,976</td>
</tr>
<tr>
<td>1,000</td>
<td>502,497</td>
<td>4,019,976</td>
</tr>
</tbody>
</table>

### Diagnostics

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

### Also see

[M-4] **Statistical** — Statistical functions