Title

stata.com

panelsetup( ) — Panel-data processing

Description

These functions assist with the processing of panel data. The idea is to make it easy and fast to write loops like

```
for (i=1; i<=number_of_panels; i++) {
      X = matrix corresponding to panel i
      ...
      ... (calculations using X) ...
      ...
}
```

Using these functions, this loop could become

```
st_view(Vid, ., "idvar",      "touse")
st_view(V,   ., ("x1", "x2"), "touse")
info = panelsetup(Vid, 1)
for (i=1; i<=rows(info); i++) {
      X = panelsubmatrix(V, i, info)
      ...
      ... (calculations using X) ...
      ...
}
```

`panelsetup(`*V*`,` *idcol*`, ...)` sets up panel processing. It returns a matrix (*info*) that is passed to other panel-processing functions.

`panelstats(`*info*`)` returns a row vector containing the number of panels, number of observations, minimum number of observations per panel, and maximum number of observations per panel.

`panelsubmatrix(`*V*`,` *i*`,` *info*`)` returns a matrix containing the contents of *V* for panel *i*.

`panelsubview(`*SV*`,` *V*`,` *i*`,` *info*`)` does nearly the same thing. Rather than returning a matrix, however, it places the matrix in *SV*. If *V* is a view, then the matrix placed in *SV* will be a view.

Syntax

info = `panelsetup(`*V*`,` *idcol*`)`

info = `panelsetup(`*V*`,` *idcol*`,` *minobs*`)`

info = `panelsetup(`*V*`,` *idcol*`,` *minobs*`,` *maxobs*`)`

real rowvector `panelstats(`*info*`)`

real matrix `panelsubmatrix(`*V*`,` *i*`,` *info*`)`

void `panelsubview(`*SV*`,` *V*`,` *i*`,` *info*`)`

where

V:	*real* or *string matrix*, possibly a view
idcol:	*real scalar*
minobs:	*real scalar*
maxobs:	*real scalar*
info:	*real matrix*
i:	*real scalar*
SV:	*matrix* to be created, possibly as view

Remarks and examples

stata.com

Remarks are presented under the following headings:

Definition of panel data

Panel data include multiple observations on subjects, countries, etc.:

Subject ID	Time ID	x1	x2
1	1	4.2	3.7
1	2	3.2	3.7
1	3	9.2	4.2
2	1	1.7	4.0
2	2	1.9	5.0
3	1	9.5	1.3
⋮	⋮	⋮	⋮

In the above dataset, there are three observations for subject 1, two for subject 2, etc. We labeled the identifier within subject to be time, but that is only suggestive, and in any case, the secondary identifier will play no role in what follows.

If we speak about the first panel, we are discussing the first 3 observations of this dataset. If we speak about the second, that corresponds to observations 4 and 5.

It is common to refer to panel numbers with the letter i. It is common to refer to the number of observations in the ith panel as T_i even when the data within panel have nothing to do with repeated observations over time.

Definition of problem

We want to calculate some statistic on panel data. The calculation amounts to

$$\sum_{i=1}^{K} f(X_i)$$

where the sum is performed across panels, and X_i is the data matrix for panel i. For instance, given the example in the previous section

$$X_1 = \begin{bmatrix} 4.2 & 3.7 \\ 3.2 & 3.7 \\ 9.2 & 4.2 \end{bmatrix}$$

and X_2 is a similarly constructed 2×2 matrix.

Depending on the nature of the calculation, there will be problems for which

1. we want to use all the panels,
2. we want to use only panels for which there are two or more observations, and
3. we want to use the same number of observations in all the panels (balanced panels).

In addition to simple problems of the sort,

$$\sum_{i=1}^{K} f(X_i)$$

you may also need to deal with problems of the form,

$$\sum_{i=1}^{K} f(X_i, Y_i, \ldots)$$

That is, you may need to deal with problems where there are multiple matrices per subject.

We use the sum operator purely for illustration, although it is the most common. Your problem might be

$$F(X_1, Y_1, \ldots, X_2, Y_2, \ldots)$$

Preparation

Before using the functions documented here, create a matrix or matrices containing the data. For illustration, it will be sufficient to create V containing all the data in our example problem:

$$V = \begin{bmatrix} 1 & 1 & 4.2 & 3.7 \\ 1 & 2 & 3.2 & 3.7 \\ 1 & 3 & 9.2 & 4.2 \\ 2 & 1 & 1.7 & 4.0 \\ 2 & 2 & 1.9 & 5.0 \\ 3 & 1 & 9.5 & 1.3 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

But you will probably find it more convenient (and we recommend) if you create at least two matrices, one containing the subject identifier and the other containing the x variables (and omit the within-subject "time" identifier altogether):

$$V1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 3 \\ \vdots \end{bmatrix} \qquad V2 = \begin{bmatrix} 4.2 & 3.7 \\ 3.2 & 3.7 \\ 9.2 & 4.2 \\ 1.7 & 4.0 \\ 1.9 & 5.0 \\ 9.5 & 1.3 \\ \vdots & \vdots \end{bmatrix}$$

In the above, matrix *V1* contains the subject identifier, and matrix *V2* contains the data for all the X_i matrices in

$$\sum_{i=1}^{K} f(X_i)$$

If your calculation is

$$\sum_{i=1}^{K} f(X_i, Y_i, \ldots)$$

create additional V matrices, *V3* corresponding to Y_i, and so on.

To create these matrices, use [M-5] **st_view( )**

```
st_view(V1,  ., "idvar",       "touse")
st_view(V2,  ., ("x1", "x2"), "touse")
```

although you could use [M-5] **st_data( )** if you preferred. Using `st_view()` will save memory. You can also construct *V1*, *V2*, ..., however you wish; they are just matrices. Be sure that the matrices align, for example, that row 4 of one matrix corresponds to row 4 of another. We did that above by assuming a *touse* variable had been included (or constructed) in the dataset.

Use of panelsetup( )

`panelsetup(`*V*`,` *idcol*`, ...)` sets up panel processing, returning a $K \times 2$ matrix that contains a row for each panel. The row records the first and last observation numbers (row numbers in V) that correspond to the panel.

For instance, with our example, `panelsetup()` will return

$$\begin{bmatrix} 1 & 3 \\ 4 & 5 \\ 6 & 9 \\ \vdots & \vdots \end{bmatrix}$$

The first panel is recorded in observations 1 to 3; it contains $3 - 1 + 1 = 3$ observations. The second panel is recorded in observations 4 to 5 and it contains $5 - 4 + 1 = 2$ observations, and so on. We recorded the third panel as being observations 6 to 9, although we did not show you enough of the original data for you to know that 9 was the last observation with ID 3.

`panelsetup()` has many more capabilities in constructing this result, but it is important to appreciate that returning this observation-number matrix is all that `panelsetup()` does. This matrix is all that other panel functions need to know. They work with the information produced by `panelsetup()`, but they will equally well work with any two-column matrix that contains observation numbers. Correspondingly, `panelsetup()` engages in no secret behavior that ties up memory, puts you in a mode, or anything else. `panelsetup()` merely produces this matrix.

The number of rows of the matrix `panelsetup()` returns equals K, the number of panels.

The syntax of `panelsetup()` is

info = `panelsetup(`*V*`,` *idcol*`,` *minobs*`,` *maxobs*`)`

The last two arguments are optional.

The required argument *V* specifies a matrix containing at least the panel identification numbers and required argument *idcol* specifies the column of *V* that contains that ID. Here we will use the matrix *V1*, which contains only the identification number:

info `= panelsetup(`*V1*`, 1)`

The two optional arguments are *minobs* and *maxobs*. *minobs* specifies the minimum number of observations within panel that we are willing to tolerate; if a panel has fewer observations, we want to omit it entirely. For instance, were we to specify

info `= panelsetup(`*V1*`, 1, 3)`

then the matrix `panelsetup()` would contain fewer rows. In our example, the returned *info* matrix would contain

$$\begin{bmatrix} 1 & 3 \\ 6 & 9 \\ \vdots & \vdots \end{bmatrix}$$

Observations 4 and 5 are now omitted because they correspond to a two-observation panel, and we said only panels with three or more observations should be included.

We chose three as a demonstration. In fact, it is most common to code

info `= panelsetup(`*V1*`, 1, 2)`

because that eliminates the singletons (panels with one observation).

The final optional argument is *maxobs*. For example,

info `= panelsetup(`*V1*`, 1, 2, 5)`

means to include only up to five observations per panel. Any observations beyond five are to be trimmed. If we code

```
info = panelsetup(V1, 1, 3, 3)
```

then all the panels contained in *info* would have three observations. If a panel had fewer than three observations, it would be omitted entirely. If a panel had more than three observations, only the first three would be included.

Panel datasets with the same number of observations per panel are said to be balanced. `panelsetup()` also provides panel-balancing capabilities. If you specify *maxobs* as 0, then

1. `panelsetup()` first calculates the $\min(T_i)$ among the panels with *minobs* observations or more. Call that number *m*.
2. `panelsetup()` then returns `panelsetup(`*V1*`, `*idcol*`, `*m*`, `*m*`)`, thus creating balanced panels of size *m* and producing a dataset that has the maximum number of within-panel observations given it has the maximum number of panels.

If we coded

```
info = panelsetup(V1, 1, 2, 0)
```

then `panelsetup()` would create the maximum number of panels with the maximum number of within-panel observations subject to the constraint of no singletons and the panels being balanced.

Using panelstats( )

`panelstats(`*info*`)` can be used on any two-column matrix that contains observation numbers. `panelstats()` returns a row vector containing

```
panelstats()[1] = number of panels (same as rows(info))
panelstats()[2] = number of observations
panelstats()[3] = min(T_i)
panelstats()[4] = max(T_i)
```

Using panelsubmatrix( )

Having created an *info* matrix using `panelsetup()`, you can obtain the matrix corresponding to the *i*th panel using

```
X = panelsubmatrix(V, i, info)
```

It is not necessary that `panelsubmatrix()` be used with the same matrix that was used to produce *info*. We created matrix *V1* containing the ID numbers, and we created matrix *V2* containing the *x* variables

```
st_view(V1,  ., "idvar",      "touse")
st_view(V2,  ., ("x1", "x2"), "touse")
```

and we create *info* using *V1*:

```
info = panelsetup(V1, 1)
```

We can now create the corresponding X matrix by coding

```
X = panelsubmatrix(V2, i, info)
```

and, had we created a *V3* matrix corresponding to Y_i, we could also code

```
Y = panelsubmatrix(V3, i, info)
```

and so on.

Using panelsubview( )

`panelsubview()` works much like `panelsubmatrix()`. The difference is that rather than coding

```
X = panelsubmatrix(V, i, info)
```

you code

```
panelsubview(X, V, i, info)
```

The matrix to be defined becomes the first argument of `panelsubview()`. That is because `panelsubview()` is designed especially to work with views. `panelsubmatrix()` will work with views, but `panelsubview()` does something special. Rather than returning an ordinary matrix (an array, in the jargon), if V is a view, `panelsubview()` returns a view in its first argument. Views save memory.

Views can save much memory, so it would seem that you would always want to use `panelsubview()` in place of `panelsubmatrix()`. What is not always appreciated, however, is that it takes Mata longer to access the data recorded in views, and so there is a tradeoff.

If the panels are likely to be large, you want to use `panelsubview()`. Conserving memory trumps all other considerations.

In fact, the panels that occur in most datasets are not that large, even when the dataset itself is. If you are going to make many calculations on X, you may wish to use `panelsubmatrix()`.

Both `panelsubmatrix()` and `panelsubview()` work with view and nonview matrices. `panelsubview()` produces a regular matrix when the base matrix V is not a view, just as does `panelsubmatrix()`. The difference is that `panelsubview()` will produce a view when V is a view, whereas `panelsubmatrix()` always produces a nonview matrix.

Conformability

`panelsetup(`*V*`, `*idcol*`, `*minobs*`, `*maxobs*`)`:

V:	$r \times c$	
idcol:	1×1	
minobs:	1×1	(optional)
maxobs:	1×1	(optional)
result:	$K \times 2$,	K = number of panels

`panelstats(`*info*`)`:

info:	$K \times 2$
result:	1×4

`panelsubmatrix(`*V*`,` *i*`,` *info*`)`:

V:	$r \times c$
i:	1×1, $1 \le i \le$ `rows(`*info*`)`
info:	$K \times 2$
result:	$t \times c$, $t =$ number of obs. in panel

`panelsubview(`*SV*`,` *V*`,` *i*`,` *info*`)`:

input:

SV:	irrelevant
V:	$r \times c$
i:	1×1, $1 \le i \le$ `rows(`*info*`)`
info:	$K \times 2$
result:	$t \times c$, $t =$ number of obs. in panel

output:

SV:	$t \times c$, $t =$ number of obs. in panel

Diagnostics

`panelsubmatrix(`*V*`,` *i*`,` *info*`)` and `panelsubview(`*SV*`,` *V*`,` *i*`,` *info*`)` abort with error if $i < 1$ or $i >$ `rows(`*info*`)`.

`panelsetup()` can return a 0×2 result.

Also see

[M-4] **Utility** — Matrix utility functions