Title

stata.com

makecns — Constrained estimation

Description Syntax Options Remarks and examples Stored results Also see

Description

`makecns` is a programmer's command that facilitates adding constraints to estimation commands.

`makecns` will create a constraint matrix and displays a note for each constraint that is dropped because of an error. When called without arguments, `makecns` will add missing factor-variable constraints implied by base levels, empty levels, and omitted coefficients. The constraint matrix is stored in `e(Cns)`.

`matcproc` returns matrices helpful for performing constrained estimation, including the constraint matrix.

If your interest is simply in using constraints in a command that supports constrained estimation, see [R] **constraint**.

Syntax

Build constraints

`makecns` [*numlist* | *matname*] [, *options*]

Create constraint matrix

`matcproc` **T a C**

numlist is a list of constraint numbers, separated by blanks or dashes; *matname* is an existing matrix representing the constraints and must have one more column than the `e(b)` and `e(V)` matrices.

T, **a**, and **C** are names of new or existing matrices.

options	Description
`nocnsnotes`	do not display notes when constraints are dropped
`displaycns`	display the system-stored constraint matrix
`r`	return the accepted constraints in `r()`; this option overrides `displaycns`

`collect` is allowed; see [U] **11.1.10 Prefix commands**.

Options

`nocnsnotes` prevents notes from being displayed when constraints are dropped.

`displaycns` displays the system-stored constraint matrix in readable form.

`r` returns the accepted constraints in `r()`. This option overrides `displaycns`.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Introduction

Users of estimation commands that allow constrained estimation define constraints with the `constraint` command; they indicate which constraints they want to use by specifying the `constraints(`*numlist*`)` option to the estimation command. This entry concerns programming such sophisticated estimators. If you are programming using `ml`, you can ignore this entry. Constraints are handled automatically (and if you were to look inside the `ml` code, you would find that it uses `makecns`).

Before reading this entry, you should be familiar with constraints from a user's perspective; see [R] **constraint**. You should also be familiar with programming estimation commands that do not include constraints; see [P] **ereturn**.

Overview

You have an estimation command and wish to allow a set of linear constraints to be specified for the parameters by the user and then to produce estimates subject to those constraints. Stata will do most of the work for you. First, it will collect the constraints—all you have to do is add an option to your estimation command to allow the user to specify which constraints to use. Second, it will process those constraints, converting them from algebraic form (such as `group1=group2`) to a constraint matrix. Third, it will convert the constraint matrix into two matrices that will, for maximum likelihood estimation, allow you to write your routine almost as if there were no constraints.

There will be a "reduced-form" parameter vector, $\mathbf{b}_c$, which your likelihood-calculation routine will receive. That vector, multiplied by one of the almost magical matrices and then added to the other, can be converted into a regular parameter vector with the constraints applied, so other than the few extra matrix calculations, you can calculate the likelihood function as if there were no constraints. You can do the same thing with respect to the first and second derivatives (if you are calculating them), except that, after getting them, you will need to perform another matrix multiplication or two to convert them into the reduced form.

Once the optimum is found, you will have reduced-form parameter vector $\mathbf{b}_c$ and variance–covariance matrix $\mathbf{V}_c$. Both can be easily converted into full-form-but-constrained $\mathbf{b}$ and $\mathbf{V}$.

Finally, you will `ereturn post` the results along with the constraint matrix Stata made up for you in the first place. You can, with a few lines of program code, arrange it so that, every time results are replayed, the constraints under which they were produced are redisplayed in standard algebraic format.

Mathematics

Let $\mathbf{R}\mathbf{b}' = \mathbf{r}$ be the constraint for $\mathbf{R}$, a $c \times p$ constraint matrix imposing c constraints on p parameters; $\mathbf{b}$, a $1 \times p$ parameter vector; and $\mathbf{r}$, a $c \times 1$ vector of constraint values.

We wish to construct a $p \times k$ matrix, $\mathbf{T}$, that takes $\mathbf{b}$ into a reduced-rank form, where $k = p - c$. There are obviously many $\mathbf{T}$ matrices that will do this; we choose one with the properties

$$\begin{aligned} \mathbf{b}_c &= \mathbf{b}_0\mathbf{T} \\ \mathbf{b} &= \mathbf{b}_c\mathbf{T}' + \mathbf{a} \end{aligned}$$

where $\mathbf{b}_c$ is a reduced-form projection of any solution $\mathbf{b}_0$; that is, $\mathbf{b}_c$ is a vector of lesser dimension ($1 \times k$ rather than $1 \times p$) that can be treated as if it were unconstrained. The second equation says that $\mathbf{b}_c$ can be mapped back into a higher-dimensioned, properly constrained $\mathbf{b}$; $1 \times p$ vector $\mathbf{a}$ is a constant that depends only on $\mathbf{R}$ and $\mathbf{r}$.

With such a $\mathbf{T}$ matrix and $\mathbf{a}$ vector, you can engage in unconstrained optimization of $\mathbf{b}_c$. If the estimate $\mathbf{b}_c$ with variance–covariance matrix $\mathbf{V}_c$ is produced, it can be mapped back into $\mathbf{b} = \mathbf{b}_c\mathbf{T}' + \mathbf{a}$ and $\mathbf{V} = \mathbf{T}\mathbf{V}_c\mathbf{T}'$. The resulting $\mathbf{b}$ and $\mathbf{V}$ can then be posted.

❑ Technical note

So how did we get so lucky? This happy solution arises if

$$\begin{aligned} \mathbf{T} &= \text{first } k \text{ eigenvectors of } \mathbf{I} - \mathbf{R}'(\mathbf{R}\mathbf{R}')^{-1}\mathbf{R} \qquad (p \times k) \\ \mathbf{L} &= \text{last } c \text{ eigenvectors of } \mathbf{I} - \mathbf{R}'(\mathbf{R}\mathbf{R}')^{-1}\mathbf{R} \qquad (p \times c) \\ \mathbf{a} &= \mathbf{r}'(\mathbf{L}'\mathbf{R}')^{-1}\mathbf{L}' \end{aligned}$$

because

$$(\mathbf{b}_c, \mathbf{r}') = \mathbf{b}(\mathbf{T}, \mathbf{R}')$$

If $\mathbf{R}$ consists of a set of consistent constraints, then it is guaranteed to have rank c. Thus $\mathbf{R}\mathbf{R}'$ is a $c \times c$ invertible matrix.

We will now show that $\mathbf{R}\mathbf{T} = \mathbf{0}$ and $\mathbf{R}(\mathbf{L}\mathbf{L}') = \mathbf{R}$.

Because $\mathbf{R}$: $c \times p$ is assumed to be of rank c, the first k eigenvalues of $\mathbf{P} = \mathbf{I} - \mathbf{R}'(\mathbf{R}\mathbf{R}')^{-1}\mathbf{R}$ are positive and the last c are zero. Break $\mathbf{R}$ into a basis spanned by these components. If $\mathbf{R}$ had any components in the first k, they could not be annihilated by $\mathbf{P}$, contradicting

$$\mathbf{R}\mathbf{P} = \mathbf{R} - \mathbf{R}\mathbf{R}'(\mathbf{R}\mathbf{R}')^{-1}\mathbf{R} = \mathbf{0}$$

Therefore, $\mathbf{T}$ and $\mathbf{R}$ are orthogonal to each other. Because $(\mathbf{T}, \mathbf{L})$ is an orthonormal basis, $(\mathbf{T}, \mathbf{L})'$ is its inverse, so $(\mathbf{T}, \mathbf{L})(\mathbf{T}, \mathbf{L})' = \mathbf{I}$. Thus

$$\begin{aligned} \mathbf{T}\mathbf{T}' + \mathbf{L}\mathbf{L}' &= \mathbf{I} \\ (\mathbf{T}\mathbf{T}' + \mathbf{L}\mathbf{L}')\mathbf{R}' &= \mathbf{R}' \\ (\mathbf{L}\mathbf{L}')\mathbf{R}' &= \mathbf{R}' \end{aligned}$$

So we conclude that $\mathbf{r} = \mathbf{b}\mathbf{R}(\mathbf{L}\mathbf{L}')$. $\mathbf{R}\mathbf{L}$ is an invertible $c \times c$ matrix, so

$$\{\mathbf{b}_c, \mathbf{r}'(\mathbf{L}'\mathbf{R}')^{-1}\} = \mathbf{b}(\mathbf{T}, \mathbf{L})$$

Remember, $(\mathbf{T},\mathbf{L})$ is a set of eigenvectors, meaning $(\mathbf{T},\mathbf{L})^{-1} = (\mathbf{T},\mathbf{L})'$, so $\mathbf{b} = \mathbf{b}_c\mathbf{T}' + \mathbf{r}'(\mathbf{L}'\mathbf{R}')^{-1}\mathbf{L}'$.

❑

If a solution is found by likelihood methods, the reduced-form parameter vector is passed to the maximizer and from there to the program that computes a likelihood value from it. To find the likelihood value, the inner routines can compute $\mathbf{b} = \mathbf{b}_c\mathbf{T}' + \mathbf{a}$. The routine may then go on to produce a set of $1 \times p$ first derivatives, $\mathbf{d}$, and $p \times p$ second derivatives, $\mathbf{H}$, even though the problem is of lesser dimension. These matrices can be reduced to the k-dimensional space via

$$\mathbf{d}_c = \mathbf{d}\mathbf{T}$$
$$\mathbf{H}_c = \mathbf{T}'\mathbf{H}\mathbf{T}$$

❑ Technical note

Alternatively, if a solution were to be found by direct matrix methods, the programmer must derive a new solution based on $\mathbf{b} = \mathbf{b}_c\mathbf{T}' + \mathbf{a}$. For example, the least-squares normal equations come from differentiating $(\mathbf{y} - \mathbf{X}\mathbf{b})^2$. Setting the derivative with respect to $\mathbf{b}$ to zero results in

$$\mathbf{T}'\mathbf{X}'\Big\{\mathbf{y} - \mathbf{X}(\mathbf{T}\mathbf{b}_c' + \mathbf{a}')\Big\} = 0$$

yielding

$$\mathbf{b}_c' = (\mathbf{T}'\mathbf{X}'\mathbf{X}\mathbf{T})^{-1}(\mathbf{T}'\mathbf{X}'\mathbf{y} - \mathbf{T}'\mathbf{X}'\mathbf{X}\mathbf{a}')$$
$$\mathbf{b}' = \mathbf{T}\Big\{(\mathbf{T}'\mathbf{X}'\mathbf{X}\mathbf{T})^{-1}(\mathbf{T}'\mathbf{X}'\mathbf{y} - \mathbf{T}'\mathbf{X}'\mathbf{X}\mathbf{a}')\Big\} + \mathbf{a}'$$

Using the matrices $\mathbf{T}$ and $\mathbf{a}$, the solution is not merely to constrain the $\mathbf{b}'$ obtained from an unconstrained solution $(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$, even though you might know that, here, with further substitutions this could be reduced to

$$\mathbf{b}' = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} + (\mathbf{X}'\mathbf{X})^{-1}\mathbf{R}'\{\mathbf{R}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{R}'\}^{-1}\{\mathbf{r} - \mathbf{R}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}\}$$

❑

Linkage of the mathematics to Stata

Users define constraints using the `constraint` command; see [R] **constraint**. The constraints are numbered, and Stata stores them in algebraic format—the same format in which the user typed them. Stata does this because, until the estimation problem is defined, it cannot know how to interpret the constraint. Think of the constraint `_b[group1]=_b[group2]`, meaning that two coefficients are to be constrained to equality, along with the constraint `_b[group3]=2`. The constraint matrices $\mathbf{R}$ and $\mathbf{r}$ are defined so that $\mathbf{R}\mathbf{b}' = \mathbf{r}$ imposes the constraint. The matrices *might* be

$$\begin{pmatrix} 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$$

if it just so happened that the third and fourth coefficients corresponded to `group1` and `group2` and the fifth corresponded to `group3`. Then again, it might look different if the coefficients were organized differently.

Therefore, Stata must wait until estimation begins to define the $\mathbf{R}$ and $\mathbf{r}$ matrices. Stata learns about the organization of a problem from the names bordering the coefficient vector and variance–covariance matrix. Therefore, Stata requires you to `ereturn post` a dummy estimation result that has the correct names. From that, it can now determine the organization of the constraint matrix and make it for you. Once an (dummy) estimation result has been posted, `makecns` can make the constraint matrices, and, once they are built, you can obtain copies of them from `e(Cns)`. Stata stores the constraint matrices $\mathbf{R}$ and $\mathbf{r}$ as a $c \times (p+1)$ matrix $\mathbf{C} = (\mathbf{R}, \mathbf{r})$. Putting them together makes it easier to pass them to subroutines.

The second step in the process is to convert the constrained problem to a reduced-form problem. We outlined the mathematics above; the `matcproc` command will produce the $\mathbf{T}$ and $\mathbf{a}$ matrices. If you are performing maximum likelihood, your likelihood, gradient, and Hessian calculation subroutines can still work in the full metric by using the same $\mathbf{T}$ and $\mathbf{a}$ matrices to translate the reduced-format parameter vector back to the original metric. If you do this, and if you are calculating gradients or Hessians, you must remember to compress them to reduced form using the $\mathbf{T}$ and $\mathbf{a}$ matrices.

When you have a reduced-form solution, you translate this back to a constrained solution using $\mathbf{T}$ and $\mathbf{a}$. You then `ereturn post` the constrained solutions, along with the original `Cns` matrix, and use `ereturn display` to display the results.

Thus the outline of a program to perform constrained estimation is

```
program myest, eclass properties(...)
        version 17.0
        if replay() {     // replay the results
                if ("`e(cmd)'" != "myest") error 301
                syntax [, Level(cilevel) ]
                makecns , displaycns
        }
        else {           // fit the model
                syntax whatever  [,                          ///
                        whatever                             ///
                        Constraints(string)                  ///
                        Level(cilevel)                       ///
                ]
                // any other parsing of the user's estimate request
                tempname b V C T a bc Vc
                local p=number of parameters
                // define the model  (set the row and column
                // names) in 'b'
                if "`constraints'" != "" {
                        matrix `V' = `b''*`b'
                        ereturn post `b' `V'          // a dummy solution
                        makecns `constraints', display
                        matcproc `T' `a' `C'
                        // obtain solution in 'bc' and 'Vc'
                        matrix `b' = `bc'*`T'' + `a'  // note prime
                        matrix `V' = `T'*`Vc'*`T''    // note prime
                        ereturn post `b' `V' `C', options
                }
                else {
                        // obtain standard solution in 'b' and 'V'
                        ereturn post `b' `V', options
                }
                // store whatever else you want in e()
```

```
                ereturn local cmd "myest"
        }
        // output any header above the coefficient table
        ereturn display, level(‘level’)
end
```

There is one point that might escape your attention: Immediately after obtaining the constraint, we display the constraints even before we undertake the estimation. This way, a user who has made a mistake may press *Break* rather than waiting until the estimation is complete to discover the error. Our code displays the constraints every time the results are reported, even when typing *myest* without arguments.

Stored results

`makecns` stores the following in `r()`:

Scalars

`r(k_autoCns)`	number of base, empty, and omitted constraints

Macros

`r(clist)`	constraints used (numlist or matrix name)

Also see

[R] **constraint** — Define and list constraints

[P] **ereturn** — Post the estimation results

[P] **macro** — Macro definition and manipulation

[P] **matrix** — Introduction to matrix commands

[P] **matrix get** — Access system matrices

[R] **cnsreg** — Constrained linear regression

[R] **ml** — Maximum likelihood estimation