# Title

stata.com

**test —** Test linear hypotheses after estimation



## Description

`test` performs Wald tests of simple and composite linear hypotheses about the parameters of the most recently fit model.

`test` supports `svy` estimators (see [SVY] **svy estimation**), carrying out an adjusted Wald test by default in such cases. `test` can be used with `svy` estimation results, see [SVY] **svy postestimation**.

`testparm` provides a useful alternative to `test` that permits *varlist* rather than a list of coefficients (which is often nothing more than a list of variables), allowing the use of standard Stata notation, including '`-`' and '`*`', which are given the *exp*ression interpretation by `test`.

`test` and `testparm` perform Wald tests. For likelihood-ratio tests, see [R] **lrtest**. For Wald-type tests of nonlinear hypotheses, see [R] **testnl**. To display estimates for one-dimensional linear or nonlinear expressions of coefficients, see [R] **lincom** and [R] **nlcom**.

See [R] **anova postestimation** for additional `test` syntax allowed after `anova`.

See [MV] **manova postestimation** for additional `test` syntax allowed after `manova`.

## Quick start

*Linear tests after single-equation models*

Joint test that the coefficients on `x1` and `x2` are equal to 0

```
test x1 x2
```

Joint test that coefficients on factor indicators `2.a` and `3.a` are equal to 0

```
test 2.a 3.a
```

Test that coefficients on indicators `2.a` and `3.a` are equal

```
test 2.a = 3.a
```

Joint test that coefficients on indicators `1.a`, `2.a`, and `3.a` are all equal

```
test (1.a=2.a) (1.a=3.a)
```

Same as above

```
test 1.a=2.a=3.a
```

As above, but add separate tests for each pairing

```
test 1.a=2.a=3.a, mtest
```

As above, but with $p$-values adjusted for multiple comparisons using Šidák's method

```
test (1.a=2.a) (1.a=3.a), mtest(sidak)
```

Test that the sum of the coefficients for x1 and x2 is equal to 4

```
test x1 + x2 = 4
```

Test the equality of two linear expressions involving coefficients on x1 and x2

```
test 2*x1 = 3*x2
```

*Shorthand varlist notation*

Joint test that all coefficients on the indicators for a are equal to 0

```
testparm i.a
```

Joint test that all coefficients on the indicators for a and b are equal to 0

```
testparm i.a i.b
```

Joint test that all coefficients associated with the interaction of factor variables a and b are equal to 0

```
testparm i.a#i.b
```

Joint test that the coefficients on all variables x* are equal to 0

```
testparm x*
```

*Linear tests after multiple-equation models*

Joint test that the coefficient on x1 is equal to 0 in all equations

```
test x1
```

Joint test that the coefficients for x1 and x2 are equal to 0 in equation y3

```
test [y3]x1 [y3]x2
```

Test that the coefficients for x1 are equal in equations y1 and y3

```
test [y1]x1=[y3]x1
```

Same as above

```
test [y1=y3]: x1
```

Joint test of the equality of coefficients for x1 and x2 across equations y1 and y3

```
test [y1=y3]: x1 x2
```

Add coefficients for x1 and x2 from equation y4 to test

```
test [y1=y3=y4]: x1 x2
```

Test that all coefficients in the equation for y1 are equal to those in the equation for y2

```
test [y1=y2]
```

As above, but only for coefficients on variables common to both equations

```
test [y1=y2], common
```

*Shorthand varlist notation*

Joint test that all coefficients on the indicators for `a` are 0 in all equations

```
testparm i.a
```

Joint test that all coefficients on the indicators for `a` are equal to each other in the first equation

```
testparm i.a, equal
```

As above, but for the equation for `y4`

```
testparm i.a, equal equation(y4)
```

Joint test that the coefficients on the indicators for `a` and `b` are equal to 0 in all equations

```
testparm i.a i.b
```

Joint test that all coefficients associated with the interaction of factors `a` and `b` are 0

```
testparm i.a#i.b
```

## Menu

Statistics > Postestimation

# Syntax

*Basic syntax*

```
test coeflist                                                    (Syntax 1)

test exp=exp[=...]                                               (Syntax 2)

test [eqno] [: coeflist]                                         (Syntax 3)

test [eqno=eqno[=...]] [: coeflist]                              (Syntax 4)

testparm varlist [, testparm_options]
```

*Full syntax*

```
test (spec) [(spec) ...] [, test_options]
```

| *testparm_options* | Description |
|---|---|
| `equal` | hypothesize that the coefficients are equal to each other |
| `equation(`*eqno*`)` | specify equation name or number for which the hypothesis is tested |
| `nosvyadjust` | compute unadjusted Wald tests for survey results |
| `df(#)` | use $F$ distribution with # denominator degrees of freedom for the reference distribution of the test statistic; for survey data, # specifies the design degrees of freedom unless `nosvyadjust` is specified |

`df(#)` does not appear in the dialog box.

| *test_options* | Description |
|---|---|
| Options | |
| `mtest[(`*opt*`)]` | test each condition separately |
| `coef` | report estimated constrained coefficients |
| `accumulate` | test hypothesis jointly with previously tested hypotheses |
| `notest` | suppress the output |
| `common` | test only variables common to all the equations |
| `constant` | include the constant in coefficients to be tested |
| `nosvyadjust` | compute unadjusted Wald tests for survey results |
| `minimum` | perform test with the constant, drop terms until the test becomes nonsingular, and test without the constant on the remaining terms; highly technical |
| `matvlc(`*matname*`)` | save the variance–covariance matrix; programmer's option |
| `df(#)` | use $F$ distribution with # denominator degrees of freedom for the reference distribution of the test statistic; for survey data, # specifies the design degrees of freedom unless `nosvyadjust` is specified |

*coeflist* and *varlist* may contain factor variables and time-series operators; see **[U] 11.4.3 Factor variables** and **[U] 11.4.4 Time-series varlists**.

`matvlc(`*matname*`)` and `df(#)` do not appear in the dialog box.

Syntax 1 tests that coefficients are 0.

Syntax 2 tests that linear expressions are equal.

Syntax 3 tests that coefficients in *eqno* are 0.

Syntax 4 tests equality of coefficients between equations.

*spec* is one of

*coeflist*
*exp*=*exp* [ =*exp* ]
[*eqno*] [ : *coeflist* ]
[*eqno*$_1$=*eqno*$_2$ [ =... ]] [ : *coeflist* ]

*coeflist* is

*coef* [*coef* ...]
[*eqno*]*coef* [ [*eqno*]*coef*... ]
[*eqno*]_b[*coef*] [ [*eqno*]_b[*coef*]... ]

*exp* is a linear expression containing

*coef*
_b[*coef*]
_b[*eqno*:*coef*]
[*eqno*]*coef*
[*eqno*]_b[*coef*]

*eqno* is

##
*name*

*coef* identifies a coefficient in the model. *coef* is typically a variable name, a level indicator, an interaction indicator, or an interaction involving continuous variables. Level indicators identify one level of a factor variable and interaction indicators identify one combination of levels of an interaction; see [U] **11.4.3 Factor variables**. *coef* may contain time-series operators; see [U] **11.4.4 Time-series varlists**.

Distinguish between `[]`, which are to be typed, and [ ], which indicate optional arguments.

Although not shown in the syntax diagram, parentheses around *spec* are required only with multiple specifications. Also, the diagram does not show that `test` may be called without arguments to redisplay the results from the last `test`.

`anova` and `manova` (see [R] **anova** and [MV] **manova**) allow the `test` syntax above plus more (see [R] **anova postestimation** for `test` after `anova`; see [MV] **manova postestimation** for `test` after `manova`).

## Options for testparm

`equal` tests that the variables appearing in *varlist*, which also appear in the previously fit model, are equal to each other rather than jointly equal to zero.

`equation(`*eqno*`)` is relevant only for multiple-equation models, such as `mvreg`, `mlogit`, and `heckman`. It specifies the equation for which the all-zero or all-equal hypothesis is tested. `equation(#1)` specifies that the test be conducted regarding the first equation `#1`. `equation(price)` specifies that the test concern the equation named `price`.

`nosvyadjust` is for use with `svy` estimation commands; see [SVY] **svy estimation**. It specifies that the Wald test be carried out without the default adjustment for the design degrees of freedom. That is, the test is carried out as $W/k \sim F(k,d)$ rather than as $(d-k+1)W/(kd) \sim F(k,d-k+1)$, where $k$ = the dimension of the test and $d$ = the total number of sampled PSUs minus the total number of strata. When the `df()` option is used, it will override the default design degrees of freedom.

The following option is available with `testparm` but is not shown in the dialog box:

`df(#)` specifies that the $F$ distribution with # denominator degrees of freedom be used for the reference distribution of the test statistic. The default is to use `e(df_r)` degrees of freedom or the chi-squared distribution if `e(df_r)` is missing. With survey data, # is the design degrees of freedom unless `nosvyadjust` is specified.

## Options for test

Options

`mtest`[`(`*opt*`)`] specifies that tests be performed for each condition separately. *opt* specifies the method for adjusting $p$-values for multiple testing. Valid values for *opt* are

| | |
|---|---|
| `bonferroni` | Bonferroni's method |
| `holm` | Holm's method |
| `sidak` | Šidák's method |
| `noadjust` | no adjustment is to be made |

Specifying `mtest` without an argument is equivalent to `mtest(noadjust)`.

`coef` specifies that the constrained coefficients be displayed.

`accumulate` allows a hypothesis to be tested jointly with the previously tested hypotheses.

`notest` suppresses the output. This option is useful when you are interested only in the joint test of several hypotheses, specified in a subsequent call of `test, accumulate`.

`common` specifies that when you use the [*eqno*$_1$`=`*eqno*$_2$[`=`...]`]` form of *spec*, the variables common to the equations *eqno*$_1$, *eqno*$_2$, etc., be tested. The default action is to complain if the equations have variables not in common.

`constant` specifies that `_cons` be included in the list of coefficients to be tested when using the [*eqno*$_1$`=`*eqno*$_2$[`=`...]`]` or [*eqno*] forms of *spec*. The default is not to include `_cons`.

`nosvyadjust` is for use with `svy` estimation commands; see [SVY] **svy estimation**. It specifies that the Wald test be carried out without the default adjustment for the design degrees of freedom. That is, the test is carried out as $W/k \sim F(k,d)$ rather than as $(d-k+1)W/(kd) \sim F(k,d-k+1)$, where $k$ = the dimension of the test and $d$ = the total number of sampled PSUs minus the total number of strata. When the `df()` option is used, it will override the default design degrees of freedom.

`minimum` is a highly technical option. It first performs the test with the constant added. If this test is singular, coefficients are dropped until the test becomes nonsingular. Then the test without the constant is performed with the remaining terms.

The following options are available with `test` but are not shown in the dialog box:

`matvlc(`*matname*`)`, a programmer's option, saves the variance–covariance matrix of the linear combinations involved in the suite of tests. For the test of the linear constraints $Lb = c$, *matname* contains $L\mathbf{V}L'$, where $\mathbf{V}$ is the estimated variance–covariance matrix of $b$.

`df(#)` specifies that the $F$ distribution with # denominator degrees of freedom be used for the reference distribution of the test statistic. The default is to use `e(df_r)` degrees of freedom or the chi-squared distribution if `e(df_r)` is missing. With survey data, # is the design degrees of freedom unless `nosvyadjust` is specified.

# Remarks and examples

stata.com

Remarks are presented under the following headings:



## Introductory examples

`test` performs $F$ or $\chi^2$ tests of linear restrictions applied to the most recently fit model (for example, `regress` or `svy: regress` in the linear regression case; `logit`, `stcox`, `svy: logit`, ... in the single-equation maximum-likelihood case; and `mlogit`, `mvreg`, `streg`, ... in the multiple-equation maximum-likelihood case). `test` may be used after *any* estimation command, although for maximum likelihood techniques, `test` produces a Wald test that depends only on the estimate of the covariance matrix—you may prefer to use the more computationally expensive likelihood-ratio test; see **[U] 20 Estimation and postestimation commands** and **[R] lrtest**.

There are several variations on the syntax for `test`. The second syntax,

`test` *exp*=*exp*[`=...`]

is allowed after any form of estimation. After fitting a model of *depvar* on `x1`, `x2`, and `x3`, typing `test x1+x2=x3` tests the restriction that the coefficients on `x1` and `x2` sum to the coefficient on `x3`. The expressions can be arbitrarily complicated; for instance, typing `test x1+2*(x2+x3)=x2+3*x3` is the same as typing `test x1+x2=x3`.

As a convenient shorthand, `test` also allows you to specify equality for multiple expressions; for example, `test x1+x2 = x3+x4 = x5+x6` tests that the three specified pairwise sums of coefficients are equal.

`test` understands that when you type `x1`, you are referring to the coefficient on `x1`. You could also more explicitly type `test _b[x1]+_b[x2]=_b[x3]`; or you could `test _coef[x1]+_coef[x2]=_coef[x3]`, or `test [#1]x1+[#1]x2=[#1]x3`, or many other things because there is more than one way to refer to an estimated coefficient; see **[U] 13.5 Accessing coefficients and standard errors**. The shorthand involves less typing. On the other hand, you must be more explicit after estimation of multiple-equation models because there may be more than one coefficient associated

with an independent variable. You might type, for instance, `test [#2]x1+[#2]x2=[#2]x3` to test the constraint in equation 2 or, more readably, `test [ford]x1+[ford]x2=[ford]x3`, meaning that Stata will test the constraint on the equation corresponding to `ford`, which might be equation 2. `ford` would be an equation name after, say, `sureg`, or, after `mlogit`, `ford` would be one of the outcomes. For `mlogit`, you could also type `test [2]x1+[2]x2=[2]x3`—note the lack of the `#`—meaning not equation 2, but the equation corresponding to the numeric outcome 2. You can even test constraints across equations: `test [ford]x1+[ford]x2=[buick]x3`.

The syntax

`test` *coeflist*

is available after all estimation commands and is a convenient way to test that multiple coefficients are zero following estimation. A *coeflist* can simply be a list of variable names,

`test` *varname* [ *varname* ... ]

and it is most often specified that way. After you have fit a model of `depvar` on `x1`, `x2`, and `x3`, typing `test x1 x3` tests that the coefficients on `x1` and `x3` are jointly zero. After multiple-equation estimation, this would test that the coefficients on `x1` and `x3` are zero in all equations that contain them. You can also be more explicit and type, for instance, `test [ford]x1 [ford]x3` to test that the coefficients on `x1` and `x3` are zero in the equation for `ford`.

In the multiple-equation case, there are more alternatives. You could also test that the coefficients on `x1` and `x3` are zero in the equation for `ford` by typing `test [ford]: x1 x3`. You could test that all coefficients except the coefficient on the constant are zero in the equation for `ford` by typing `test [ford]`. You could test that the coefficients on `x1` and `x3` in the equation for `ford` are equal to the corresponding coefficients in the equation corresponding to `buick` by typing `test[ford=buick]: x1 x3`. You could test that all the corresponding coefficients except the constant in three equations are equal by typing `test [ford=buick=volvo]`.

`testparm` is much like the first syntax of `test`. Its usefulness will be demonstrated below.

The examples below use `regress`, but what is said applies equally after any single-equation estimation command (such as `logistic`). It also applies after multiple-equation estimation commands as long as references to coefficients are qualified with an equation name or number in square brackets placed before them. The convenient syntaxes for dealing with tests of many coefficients in multiple-equation models are demonstrated in *Special syntaxes after multiple-equation estimation* below.

## ▷ Example 1: Testing for a single coefficient against zero

We have 1980 census data on the 50 states recording the birth rate in each state (`brate`), the median age (`medage`), and the region of the country in which each state is located.

The `region` variable is 1 if the state is in the Northeast, 2 if the state is in the North Central, 3 if the state is in the South, and 4 if the state is in the West. We estimate the following regression:

```
. use http://www.stata-press.com/data/r15/census3
(1980 Census data by state)
. regress brate medage c.medage#c.medage i.region
```

| Source | SS | df | MS | Number of obs | = | 50 |
|---|---|---|---|---|---|---|
| | | | | F(5, 44) | = | 100.63 |
| Model | 38803.4208 | 5 | 7760.68416 | Prob > F | = | 0.0000 |
| Residual | 3393.39921 | 44 | 77.1227094 | R-squared | = | 0.9196 |
| | | | | Adj R-squared | = | 0.9104 |
| Total | 42196.82 | 49 | 861.159592 | Root MSE | = | 8.782 |

| brate | Coef. | Std. Err. | t | P>\|t\| | [95% Conf. | Interval] |
|---|---|---|---|---|---|---|
| medage | -109.0958 | 13.52452 | -8.07 | 0.000 | -136.3527 | -81.83892 |
| c.medage# c.medage | 1.635209 | .2290536 | 7.14 | 0.000 | 1.173582 | 2.096836 |
| region | | | | | | |
| NCentral | 15.00283 | 4.252067 | 3.53 | 0.001 | 6.433353 | 23.57231 |
| South | 7.366445 | 3.953335 | 1.86 | 0.069 | -.6009775 | 15.33387 |
| West | 21.39679 | 4.650601 | 4.60 | 0.000 | 12.02412 | 30.76946 |
| _cons | 1947.611 | 199.8405 | 9.75 | 0.000 | 1544.859 | 2350.363 |

`test` can now be used to perform a variety of statistical tests. Specify the `coeflegend` option with your estimation command to see a legend of the coefficients and how to specify them; see [R] **estimation options**. We can test the hypothesis that the coefficient on `3.region` is zero by typing

```
. test 3.region=0
 ( 1)  3.region = 0
       F(  1,    44) =    3.47
            Prob > F =    0.0691
```

The $F$ statistic with 1 numerator and 44 denominator degrees of freedom is 3.47. The significance level of the test is 6.91%—we can reject the hypothesis at the 10% level but not at the 5% level.

This result from `test` is identical to one presented in the output from `regress`, which indicates that the $t$ statistic on the `3.region` coefficient is 1.863 and that its significance level is 0.069. The $t$ statistic presented in the output can be used to test the hypothesis that the corresponding coefficient is zero, although it states the test in slightly different terms. The $F$ distribution with 1 numerator degree of freedom is, however, identical to the $t^2$ distribution. We note that $1.863^2 \approx 3.47$ and that the significance levels in each test agree, although one extra digit is presented by the `test` command.

◁

## ❑ Technical note

After all estimation commands, including those that use the maximum likelihood method, the test that one variable is zero is identical to that reported by the command's output. The tests are performed in the same way—using the estimated covariance matrix—and are known as Wald tests. If the estimation command reports significance levels and confidence intervals using $z$ rather than $t$ statistics, `test` reports results using the $\chi^2$ rather than the $F$ statistic.

❑

## ▷ Example 2: Testing the value of a single coefficient

If that were all `test` could do, it would be useless. We can use `test`, however, to perform other tests. For instance, we can `test` the hypothesis that the coefficient on `2.region` is 21 by typing

```
. test 2.region=21
 ( 1)  2.region = 21
       F(  1,    44) =    1.99
            Prob > F =    0.1654
```

We find that we cannot reject that hypothesis, or at least we cannot reject it at any significance level below 16.5%.

◁

## ▷ Example 3: Testing the equality of two coefficients

The previous test is useful, but we could almost as easily perform it by hand using the results presented in the regression output if we were well read on our statistics. We could type

```
. display Ftail(1,44,((_coef[2.region]-21)/4.252068)^2)
.16544873
```

So, now let's `test` something a bit more difficult: whether the coefficient on `2.region` is the same as the coefficient on `4.region`:

```
. test 2.region=4.region
 ( 1)  2.region - 4.region = 0
       F(  1,    44) =    2.84
            Prob > F =    0.0989
```

We find that we cannot reject the equality hypothesis at the 5% level, but we can at the 10% level.

◁

## ▷ Example 4

When we tested the equality of the `2.region` and `4.region` coefficients, Stata rearranged our algebra. When Stata displayed its interpretation of the specified test, it indicated that we were testing whether `2.region` *minus* `4.region` is zero. The rearrangement is innocuous and, in fact, allows Stata to perform much more complicated algebra, for instance,

```
. test 2*(2.region-3*(3.region-4.region))=3.region+2.region+6*(4.region-3.region)
 ( 1)  2.region - 3.region = 0
       F(  1,    44) =    5.06
            Prob > F =    0.0295
```

Although we requested what appeared to be a lengthy hypothesis, once Stata simplified the algebra, it realized that all we wanted to do was test whether the coefficient on `2.region` is the same as the coefficient on `3.region`.

◁

## ❑ Technical note

Stata's ability to simplify and test complex hypotheses is limited to *linear* hypotheses. If you attempt to `test` a nonlinear hypothesis, you will be told that it is not possible:

```
. test 2.region/3.region=2.region+3.region
not possible with test
r(131);
```

To test a nonlinear hypothesis, see [R] **testnl**.

❑

## ▷ Example 5: Testing joint hypotheses

The real power of `test` is demonstrated when we test *joint* hypotheses. Perhaps we wish to test whether the region variables, taken as a whole, are significant by testing whether the coefficients on `2.region`, `3.region`, and `4.region` are simultaneously zero. `test` allows us to specify multiple conditions to be tested, each embedded within parentheses.

```
. test (2.region=0) (3.region=0) (4.region=0)
 ( 1)  2.region = 0
 ( 2)  3.region = 0
 ( 3)  4.region = 0
       F(  3,    44) =    8.85
            Prob > F =    0.0001
```

`test` displays the set of conditions and reports an $F$ statistic of 8.85. `test` also reports the degrees of freedom of the test to be 3, the "dimension" of the hypothesis, and the residual degrees of freedom, 44. The significance level of the test is close to 0, so we can strongly reject the hypothesis of no difference between the regions.

An alternative method to specify simultaneous hypotheses uses the convenient shorthand of conditions with multiple equality operators.

```
. test 2.region=3.region=4.region=0
 ( 1)  2.region - 3.region = 0
 ( 2)  2.region - 4.region = 0
 ( 3)  2.region = 0
       F(  3,    44) =    8.85
            Prob > F =    0.0001
```

◁

## ❑ Technical note

Another method to test simultaneous hypotheses is to specify a `test` for each constraint and `accumulate` it with the previous constraints:

```
. test 2.region=0
 ( 1)  2.region = 0
       F(  1,    44) =   12.45
            Prob > F =    0.0010
. test 3.region=0, accumulate
 ( 1)  2.region = 0
 ( 2)  3.region = 0
       F(  2,    44) =    6.42
            Prob > F =    0.0036
. test 4.region=0, accumulate
 ( 1)  2.region = 0
 ( 2)  3.region = 0
 ( 3)  4.region = 0
       F(  3,    44) =    8.85
            Prob > F =    0.0001
```

We tested the hypothesis that the coefficient on `2.region` was zero by typing `test 2.region=0`. We then tested whether the coefficient on `3.region` was also zero by typing `test 3.region=0, accumulate`. The `accumulate` option told Stata that this was not the start of a new test but a continuation of a previous one. Stata responded by showing us the two equations and reporting an $F$ statistic of 6.42. The significance level associated with those two coefficients being zero is 0.36%.

When we added the last constraint `test 4.region=0, accumulate`, we discovered that the three region variables are significant. If all we wanted was the overall significance and we did not want to bother seeing the interim results, we could have used the `notest` option:

```
. test 2.region=0, notest
 ( 1)  2.region = 0
. test 3.region=0, accumulate notest
 ( 1)  2.region = 0
 ( 2)  3.region = 0
. test 4.region=0, accumulate
 ( 1)  2.region = 0
 ( 2)  3.region = 0
 ( 3)  4.region = 0
       F(  3,    44) =    8.85
            Prob > F =    0.0001
```

❑

## ▷ Example 6: Quickly testing coefficients against zero

Because tests that coefficients are zero are so common in applied statistics, the `test` command has a more convenient syntax to accommodate this case:

```
. test 2.region 3.region 4.region
 ( 1)  2.region = 0
 ( 2)  3.region = 0
 ( 3)  4.region = 0
       F(  3,    44) =    8.85
            Prob > F =    0.0001
```

◁

## ▷ Example 7: Specifying varlists

We will now show how to use `testparm`. In its first syntax, `test` accepts a list of variable names but not a *varlist*.

```
. test i(2/4).region
i not found
r(111);
```

In the varlist, `i(2/4).region` means all the level variables from `2.region` through `4.region`, yet we received an error. `test` does not actually understand varlists, but `testparm` does. In fact, it understands only varlists.

```
. testparm i(2/4).region
 ( 1)  2.region = 0
 ( 2)  3.region = 0
 ( 3)  4.region = 0
       F(  3,    44) =    8.85
            Prob > F =    0.0001
```

Another way to test all the `region` variables is to type `testparm i.region`.

That `testparm` accepts varlists has other advantages that do not involve factor variables. Suppose that we have a dataset that has dummy variables `reg2`, `reg3`, and `reg4`, rather than the categorical variable `region`.

```
. use http://www.stata-press.com/data/r15/census4
(birth rate, median age)
. regress brate medage c.medage#c.medage reg2 reg3 reg4
  (output omitted)
. test reg2-reg4
- not found
r(111);
```

In a varlist, `reg2-reg4` means variables `reg2` and `reg4` and all the variables between, yet we received an error. `test` is confused because the `-` has two meanings: it means subtraction in an expression and "through" in a *varlist*. Similarly, '`*`' means "any set of characters" in a *varlist* and multiplication in an expression. `testparm` avoids this confusion—it allows only a *varlist*.

```
. testparm reg2-reg4
 ( 1)  reg2 = 0
 ( 2)  reg3 = 0
 ( 3)  reg4 = 0
       F(  3,    44) =    8.85
            Prob > F =    0.0001
```

`testparm` has another advantage. We have five variables in our dataset that start with the characters `reg`: `region`, `reg1`, `reg2`, `reg3`, and `reg4`. `reg*` thus means those five variables:

```
. describe reg*
              storage   display    value
variable name   type    format     label      variable label
------------------------------------------------------------------------------
region          int     %8.0g      region     Census Region
reg1            byte    %9.0g                 region==NE
reg2            byte    %9.0g                 region==N Cntrl
reg3            byte    %9.0g                 region==South
reg4            byte    %9.0g                 region==West
```

We cannot type `test reg*` because, in an expression, '`*`' means multiplication, but here is what would happen if we attempted to test all the variables that begin with `reg`:

```
. test region reg1 reg2 reg3 reg4
region not found
r(111);
```

The variable `region` was not included in our model, so it was not found. However, with `testparm`,

```
. testparm reg*
 ( 1)  reg2 = 0
 ( 2)  reg3 = 0
 ( 3)  reg4 = 0
       F(  3,    44) =    8.85
            Prob > F =    0.0001
```

That is, `testparm` took `reg*` to mean all variables that start with `reg` that were in our model.

◁

## ❑ Technical note

Actually, `reg*` means what it always does—all variables in our dataset that begin with `reg`—in this case, `region reg1 reg2 reg3 reg4`. `testparm` just ignores any variables you specify that are not in the model.

❑

## ▷ Example 8: Replaying the previous test

We just used `test` (`testparm`, actually, but it does not matter) to test the hypothesis that `reg2`, `reg3`, and `reg4` are jointly zero. We can review the results of our last test by typing `test` without arguments:

```
. test
 ( 1)  reg2 = 0
 ( 2)  reg3 = 0
 ( 3)  reg4 = 0
       F(  3,    44) =    8.85
            Prob > F =    0.0001
```

◁

## ❑ Technical note

`test` does not care how we build joint hypotheses; we may freely mix different forms of syntax. (We can even start with `testparm`, but we cannot use it thereafter because it does not have an `accumulate` option.)

Say that we type `test reg2 reg3 reg4` to test that the coefficients on our region dummies are jointly zero. We could then add a fourth constraint, say, that `medage` = 100, by typing `test medage=100, accumulate`. Or, if we had introduced the `medage` constraint first (our first `test` command had been `test medage=100`), we could then add the region dummy test by typing `test reg2 reg3 reg4, accumulate` or `test (reg2=0) (reg3=0) (reg4=0), accumulate`.

Remember that all previous tests are cleared when we do not specify the `accumulate` option. No matter what tests we performed in the past, if we type `test medage c.medage#c.medage`, omitting the `accumulate` option, we would test that `medage` and `c.medage#c.medage` are jointly zero.

❑

## ▷ Example 9: Testing the equality of multiple coefficients

Let's return to our `census3.dta` dataset and test the hypothesis that all the included regions have the *same* coefficient—that the Northeast is significantly different from the rest of the nation:

```
. use http://www.stata-press.com/data/r15/census3
(1980 Census data by state)
. regress brate medage c.medage#c.medage i.region
  (output omitted)
. test 2.region=3.region=4.region
 ( 1)  2.region - 3.region = 0
 ( 2)  2.region - 4.region = 0
       F(  2,    44) =    8.23
            Prob > F =    0.0009
```

We find that they are not all the same. The syntax `2.region=3.region=4.region` with multiple `=` operators is just a convenient shorthand for typing that the first expression equals the second expression and that the first expression equals the third expression,

```
. test (2.region=3.region) (2.region=4.region)
```

We performed the test for equality of the three regions by imposing two constraints: region 2 has the same coefficient as region 3, and region 2 has the same coefficient as region 4. Alternatively, we could have tested that the coefficients on regions 2 and 3 are the same and that the coefficients on regions 3 and 4 are the same. We would obtain the same results in either case.

To test for equality of the three regions, we might, likely by mistake, type equality constraints for *all* pairs of regions:

```
. test (2.region=3.region) (2.region=4.region) (3.region=4.region)
 ( 1)  2.region - 3.region = 0
 ( 2)  2.region - 4.region = 0
 ( 3)  3.region - 4.region = 0
       Constraint 3 dropped
       F(  2,    44) =    8.23
            Prob > F =    0.0009
```

Equality of regions 2 and 3 and of regions 2 and 4, however, implies equality of regions 3 and 4. `test` recognized that the last constraint is implied by the other constraints and hence dropped it.

◁

## ❑ Technical note

Generally, Stata uses `=` for assignment, as in `gen` *newvar* `=` *exp*, and `==` as the operator for testing equality in expressions. For your convenience, `test` allows both `=` and `==` to be used.

❑

## ▷ Example 10

The test for the equality of the regions is also possible with the `testparm` command. When we include the `equal` option, `testparm` tests that the coefficients of all the variables specified are equal:

```
. testparm i(2/4).region, equal
 ( 1)  - 2.region + 3.region = 0
 ( 2)  - 2.region + 4.region = 0
       F(  2,    44) =    8.23
            Prob > F =    0.0009
```

We can also obtain the equality test by accumulating single equality tests.

```
. test 2.region=3.region, notest
 ( 1)  2.region - 3.region = 0
. test 2.region=4.region, accum
 ( 1)  2.region - 3.region = 0
 ( 2)  2.region - 4.region = 0
       F(  2,    44) =    8.23
            Prob > F =    0.0009
```

◁

## ❑ Technical note

If we specify a set of inconsistent constraints, `test` will tell us by dropping the constraint or constraints that led to the inconsistency. For instance, let's `test` that the coefficients on region 2 and region 4 are the same, add the test that the coefficient on region 2 is 20, and finally add the test that the coefficient on region 4 is 21:

```
. test (2.region=4.region) (2.region=20) (4.region=21)
 ( 1)  2.region - 4.region = 0
 ( 2)  2.region = 20
 ( 3)  4.region = 21
       Constraint 2 dropped
       F(  2,    44) =    1.82
            Prob > F =    0.1737
```

`test` informed us that it was dropping constraint 2. All three equations cannot be simultaneously true, so `test` drops whatever it takes to get back to something that makes sense.

❑

## Special syntaxes after multiple-equation estimation

Everything said above about tests after single-equation estimation applies to tests after multiple-equation estimation, as long as you remember to specify the equation name. To demonstrate, let's estimate a seemingly unrelated regression by using `sureg`; see [R] **sureg**.

```
. use http://www.stata-press.com/data/r15/auto
(1978 Automobile Data)
. sureg (price foreign mpg displ) (weight foreign length)
Seemingly unrelated regression
```

| Equation | Obs | Parms | RMSE | "R-sq" | chi2 | P |
|---|---|---|---|---|---|---|
| price | 74 | 3 | 2165.321 | 0.4537 | 49.64 | 0.0000 |
| weight | 74 | 2 | 245.2916 | 0.8990 | 661.84 | 0.0000 |

| | Coef. | Std. Err. | z | P>\|z\| | [95% Conf. | Interval] |
|---|---|---|---|---|---|---|
| price | | | | | | |
| foreign | 3058.25 | 685.7357 | 4.46 | 0.000 | 1714.233 | 4402.267 |
| mpg | -104.9591 | 58.47209 | -1.80 | 0.073 | -219.5623 | 9.644042 |
| displacement | 18.18098 | 4.286372 | 4.24 | 0.000 | 9.779842 | 26.58211 |
| _cons | 3904.336 | 1966.521 | 1.99 | 0.047 | 50.0263 | 7758.645 |
| weight | | | | | | |
| foreign | -147.3481 | 75.44314 | -1.95 | 0.051 | -295.2139 | .517755 |
| length | 30.94905 | 1.539895 | 20.10 | 0.000 | 27.93091 | 33.96718 |
| _cons | -2753.064 | 303.9336 | -9.06 | 0.000 | -3348.763 | -2157.365 |

To test the significance of `foreign` in the `price` equation, we could type

```
. test [price]foreign
 ( 1)  [price]foreign = 0
           chi2(  1) =   19.89
         Prob > chi2 =    0.0000
```

which is the same result reported by `sureg`: $4.460^2 \approx 19.89$. To test `foreign` in both equations, we could type

```
. test [price]foreign [weight]foreign
 ( 1)  [price]foreign = 0
 ( 2)  [weight]foreign = 0
           chi2(  2) =   31.61
         Prob > chi2 =    0.0000
```

or

```
. test foreign
 ( 1)  [price]foreign = 0
 ( 2)  [weight]foreign = 0
           chi2(  2) =   31.61
         Prob > chi2 =    0.0000
```

This last syntax—typing the variable name by itself—tests the coefficients in all equations in which they appear. The variable `length` appears in only the `weight` equation, so typing

```
. test length
 ( 1)  [weight]length = 0
           chi2(  1) =  403.94
         Prob > chi2 =    0.0000
```

yields the same result as typing `test [weight]length`. We may also specify a linear expression rather than a list of coefficients:

```
. test mpg=displ
 ( 1)  [price]mpg - [price]displacement = 0
           chi2(  1) =    4.85
         Prob > chi2 =    0.0277
```

or

```
. test [price]mpg = [price]displ
 ( 1)  [price]mpg - [price]displacement = 0
           chi2(  1) =    4.85
         Prob > chi2 =    0.0277
```

A variation on this syntax can be used to test cross-equation constraints:

```
. test [price]foreign = [weight]foreign
 ( 1)  [price]foreign - [weight]foreign = 0
           chi2(  1) =   23.07
         Prob > chi2 =    0.0000
```

Typing an equation name in square brackets by itself tests all the coefficients except the intercept in that equation:

```
. test [price]
 ( 1)  [price]foreign = 0
 ( 2)  [price]mpg = 0
 ( 3)  [price]displacement = 0
           chi2(  3) =   49.64
         Prob > chi2 =    0.0000
```

Typing an equation name in square brackets, a colon, and a list of variable names tests those variables in the specified equation:

```
. test [price]: foreign displ
 ( 1)  [price]foreign = 0
 ( 2)  [price]displacement = 0
           chi2(  2) =   25.19
         Prob > chi2 =    0.0000
```

`test` [*eqname*$_1$=*eqname*$_2$] tests that all the coefficients in the two equations are equal. We cannot use that syntax here because there are different variables in the model:

```
. test [price=weight]
variables differ between equations
(to test equality of coefficients in common, specify option common)
r(111);
```

The `common` option specifies a test of the equality coefficients common to the equations `price` and `weight`,

```
. test [price=weight], common
 ( 1)  [price]foreign - [weight]foreign = 0
           chi2(  1) =   23.07
         Prob > chi2 =    0.0000
```

By default, `test` does not include the constant, the coefficient of the constant variable `_cons`, in the test. The `cons` option specifies that the constant be included.

```
. test [price=weight], common cons
 ( 1)  [price]foreign - [weight]foreign = 0
 ( 2)  [price]_cons - [weight]_cons = 0
           chi2(  2) =   51.23
         Prob > chi2 =    0.0000
```

We can also use a modification of this syntax with the model if we also type a colon and the names of the variables we want to test:

```
. test [price=weight]: foreign
 ( 1)  [price]foreign - [weight]foreign = 0
           chi2(  1) =   23.07
         Prob > chi2 =    0.0000
```

We have only one variable in common between the two equations, but if there had been more, we could have listed them.

Finally, a simultaneous test of multiple constraints may be specified just as after single-equation estimation.

```
. test ([price]: foreign) ([weight]: foreign)
 ( 1)  [price]foreign = 0
 ( 2)  [weight]foreign = 0
           chi2(  2) =   31.61
         Prob > chi2 =    0.0000
```

`test` can also test for equality of coefficients across more than two equations. For instance, `test [eq1=eq2=eq3]` specifies a test that the coefficients in the three equations `eq1`, `eq2`, and `eq3` are equal. This requires that the same variables be included in the three equations. If some variables are entered only in some of the equations, you can type `test [eq1=eq2=eq3], common` to test that the coefficients of the variables common to all three equations are equal. Alternatively, you can explicitly list the variables for which equality of coefficients across the equations is to be tested. For instance, `test [eq1=eq2=eq3]: time money` tests that the coefficients of the variables `time` and `money` do not differ between the equations.

## ❑ Technical note

`test [eq1=eq2=eq3], common` tests the equality of the coefficients common to all equations, but it does *not* test the equality of all common coefficients. Consider the case where

| | |
|---|---|
| `eq1` | contains the variables `var1 var2 var3` |
| `eq2` | contains the variables `var1 var2 var4` |
| `eq3` | contains the variables `var1 var3 var4` |

Obviously, only `var1` is common to all three equations. Thus `test [eq1=eq2=eq3], common` tests that the coefficients of `var1` do not vary across the equations, so it is equivalent to `test [eq1=eq2=eq3]: var1`. To perform a test of the coefficients of variables common to two equations, you could explicitly list the constraints to be tested,

```
. test ([eq1=eq2=eq3]:var1) ([eq1=eq2]:var2) ([eq1=eq3]:var3) ([eq2=eq3]:var4)
```

or use `test` with the `accumulate` option, and maybe also with the `notest` option, to form the appropriate joint hypothesis:

```
. test [eq1=eq2], common notest
. test [eq1=eq3], common accumulate notest
. test [eq2=eq3], common accumulate
```

❑

## Constrained coefficients

If the test indicates that the data do not allow you to conclude that the constraints are not satisfied, you may want to inspect the constrained coefficients. The `coef` option specified that the constrained results, estimated by GLS, are shown.

```
. test [price=weight], common coef
 ( 1)  [price]foreign - [weight]foreign = 0
           chi2(  1) =   23.07
         Prob > chi2 =    0.0000

Constrained coefficients
```

| | Coef. | Std. Err. | z | P>\|z\| | [95% Conf. | Interval] |
|---|---|---|---|---|---|---|
| price | | | | | | |
| foreign | -216.4015 | 74.06083 | -2.92 | 0.003 | -361.558 | -71.2449 |
| mpg | -121.5717 | 58.36972 | -2.08 | 0.037 | -235.9742 | -7.169116 |
| displacement | 7.632566 | 3.681114 | 2.07 | 0.038 | .4177148 | 14.84742 |
| _cons | 7312.856 | 1834.034 | 3.99 | 0.000 | 3718.215 | 10907.5 |
| weight | | | | | | |
| foreign | -216.4015 | 74.06083 | -2.92 | 0.003 | -361.558 | -71.2449 |
| length | 30.34875 | 1.534815 | 19.77 | 0.000 | 27.34057 | 33.35693 |
| _cons | -2619.719 | 302.6632 | -8.66 | 0.000 | -3212.928 | -2026.51 |

The constrained coefficient of `foreign` is $-216.40$ with standard error 74.06 in equations `price` and `weight`. The other coefficients and their standard errors are affected by imposing the equality constraint of the two coefficients of `foreign` because the unconstrained estimates of these two coefficients were correlated with the estimates of the other coefficients.

### ❑ Technical note

The two-step constrained coefficients $b_c$ displayed by `test, coef` are asymptotically equivalent to the one-stage constrained estimates that are computed by specifying the constraints during estimation using the `constraint()` option of estimation commands (Gourieroux and Monfort 1995, chap. 10). Generally, one-step constrained estimates have better small-sample properties. For inspection and interpretation, however, two-step constrained estimates are a convenient alternative. Moreover, some estimation commands (for example, `stcox`, many `xt` estimators) do not have a `constraint()` option.

❑

## Multiple testing

When performing the test of a joint hypothesis, you might want to inspect the underlying 1-degree-of-freedom hypotheses. Which constraint "is to blame"? `test` displays the univariate as well as the simultaneous test if the `mtest` option is specified. For example,

```
. test [price=weight], common cons mtest
 ( 1)  [price]foreign - [weight]foreign = 0
 ( 2)  [price]_cons - [weight]_cons = 0

------------------------------------------------------------------------------
             |        chi2     df       p
-------------+----------------------------------------------------------------
         (1) |       23.07      1     0.0000 #
         (2) |       11.17      1     0.0008 #
-------------+----------------------------------------------------------------
         all |       51.23      2     0.0000
------------------------------------------------------------------------------
                                  # unadjusted p-values
```

Both coefficients seem to contribute to the highly significant result. The 1-degree-of-freedom test shown here is identical to those if `test` had been invoked to test just this simple hypotheses. There is, of course, a real risk in inspecting these simple hypotheses. Especially in high-dimensional hypotheses, you may easily find one hypothesis that happens to be significant. Multiple testing procedures are designed to provide some safeguard against this risk. $p$-values of the univariate hypotheses are modified so that the probability of falsely rejecting one of the null hypotheses is bounded. `test` provides the methods based on Bonferroni, Šidák, and Holm.

```
. test [price=weight], common cons mtest(b)
 ( 1)  [price]foreign - [weight]foreign = 0
 ( 2)  [price]_cons - [weight]_cons = 0

------------------------------------------------------------------------------
             |        chi2     df       p
-------------+----------------------------------------------------------------
         (1) |       23.07      1     0.0000 #
         (2) |       11.17      1     0.0017 #
-------------+----------------------------------------------------------------
         all |       51.23      2     0.0000
------------------------------------------------------------------------------
                          # Bonferroni-adjusted p-values
```

# Stored results

`test` and `testparm` store the following in `r()`:

Scalars

| | | | |
|---|---|---|---|
| `r(p)` | two-sided $p$-value | `r(chi2)` | $\chi^2$ |
| `r(F)` | $F$ statistic | `r(ss)` | sum of squares (test) |
| `r(df)` | test constraints degrees of freedom | `r(rss)` | residual sum of squares |
| `r(df_r)` | residual degrees of freedom | `r(drop)` | 1 if constraints were dropped, 0 otherwise |
| `r(dropped_i)` | index of $i$th constraint dropped | | |

Macros

| | |
|---|---|
| `r(mtmethod)` | method of adjustment for multiple testing |

Matrices

| | |
|---|---|
| `r(mtest)` | multiple test results |

`r(ss)` and `r(rss)` are defined only when `test` is used for testing effects after `anova`.

## Methods and formulas

`test` and `testparm` perform Wald tests. Let the estimated coefficient vector be $\mathbf{b}$ and the estimated variance–covariance matrix be $\mathbf{V}$. Let $\mathbf{Rb} = \mathbf{r}$ denote the set of $q$ linear hypotheses to be tested jointly.

The Wald test statistic is (Judge et al. 1985, 20–28)

$$W = (\mathbf{Rb} - \mathbf{r})'(\mathbf{RVR}')^{-1}(\mathbf{Rb} - \mathbf{r})$$

If the estimation command reports its significance levels using $Z$ statistics, a chi-squared distribution with $q$ degrees of freedom,

$$W \sim \chi^2_q$$

is used for computation of the significance level of the hypothesis test.

If the estimation command reports its significance levels using $t$ statistics with $d$ degrees of freedom, an $F$ statistic,

$$F = \frac{1}{q}W$$

is computed, and an $F$ distribution with $q$ numerator degrees of freedom and $d$ denominator degrees of freedom computes the significance level of the hypothesis test.

The two-step constrained estimates $b_c$ displayed by `test` with the `coef` option are the GLS estimates of the unconstrained estimates $b$ subject to the specified constraints $Rb = c$ (Gourieroux and Monfort 1995, chap. 10),

$$\mathbf{b_c} = \mathbf{b} - \mathbf{VR}'(\mathbf{RVR}')^{-1}(\mathbf{Rb} - \mathbf{r})$$

with variance–covariance matrix

$$\mathbf{V_c} = \mathbf{V} - \mathbf{VR}'(\mathbf{RVR}')^{-1}\mathbf{RV}$$

If `test` displays a Wald test for joint (simultaneous) hypotheses, it can also display all 1-degree-of-freedom tests, with $p$-values adjusted for multiple testing. Let $p_1, p_2, \ldots, p_k$ be the unadjusted $p$-values of these 1-degree-of-freedom tests. The Bonferroni-adjusted $p$-values are defined as $p^b_i = \min(1, kp_i)$. The Šidák-adjusted $p$-values are $p^s_i = 1 - (1 - p_i)^k$. Holm's method for adjusting $p$-values is defined as $p^h_i = \min(1, k_i p_i)$, where $k_i$ is the number of $p$-values at least as large as $p_i$. Note that $p^h_i < p^b_i$, reflecting that Holm's method is strictly less conservative than the widely used Bonferroni method.

If `test` is used after a `svy` command, it carries out an adjusted Wald test—this adjustment should not be confused with the adjustment for multiple testing. Both adjustments may actually be combined. Specifically, the survey adjustment uses an approximate $F$ statistic $(d-k+1)W/(kd)$, where $W$ is the Wald test statistic, $k$ is the dimension of the hypothesis test, and $d$ = the total number of sampled PSUs minus the total number of strata. Under the null hypothesis, $(d-k+1)F/(kd) \sim F(k, d-k+1)$, where $F(k, d-k+1)$ is an $F$ distribution with $k$ numerator degrees of freedom and $d-k+1$ denominator degrees of freedom. If `nosvyadjust` is specified, the $p$-value is computed using $W/k \sim F(k, d)$.

See Korn and Graubard (1990) for a detailed description of the Bonferroni adjustment technique and for a discussion of the relative merits of it and of the adjusted and unadjusted Wald tests.

## Acknowledgment

The svy adjustment code was adopted from another command developed in collaboration with John L. Eltinge of the Bureau of Labor Statistics.

## References

Beale, E. M. L. 1960. Confidence regions in non-linear estimation. *Journal of the Royal Statistical Society, Series B* 22: 41–88.

Canette, I. 2014. Using gsem to combine estimation results. *The Stata Blog: Not Elsewhere Classified.* http://blog.stata.com/2014/08/18/using-gsem-to-combine-estimation-results/.

Eltinge, J. L., and W. M. Sribney. 1996. svy5: Estimates of linear combinations and hypothesis tests for survey data. *Stata Technical Bulletin* 31: 31–42. Reprinted in *Stata Technical Bulletin Reprints*, vol. 6, pp. 246–259. College Station, TX: Stata Press.

Gourieroux, C. S., and A. Monfort. 1995. *Statistics and Econometric Models, Vol 1: General Concepts, Estimation, Prediction, and Algorithms.* Trans. Q. Vuong. Cambridge: Cambridge University Press.

Holm, S. 1979. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics* 6: 65–70.

Judge, G. G., W. E. Griffiths, R. C. Hill, H. Lütkepohl, and T.-C. Lee. 1985. *The Theory and Practice of Econometrics.* 2nd ed. New York: Wiley.

Korn, E. L., and B. I. Graubard. 1990. Simultaneous testing of regression coefficients with complex survey data: Use of Bonferroni $t$ statistics. *American Statistician* 44: 270–276.

Mehmetoglu, M., and T. G. Jakobsen. 2017. *Applied Statistics Using Stata: A Guide for the Social Sciences.* Thousand Oaks, CA: Sage.

Weesie, J. 1999. sg100: Two-stage linear constrained estimation. *Stata Technical Bulletin* 47: 24–30. Reprinted in *Stata Technical Bulletin Reprints*, vol. 8, pp. 217–225. College Station, TX: Stata Press.

## Also see

[R] **anova** — Analysis of variance and covariance

[R] **anova postestimation** — Postestimation tools for anova

[R] **contrast** — Contrasts and linear hypothesis tests after estimation

[R] **lincom** — Linear combinations of parameters

[R] **lrtest** — Likelihood-ratio test after estimation

[R] **nestreg** — Nested model statistics

[R] **nlcom** — Nonlinear combinations of estimators

[R] **testnl** — Test nonlinear hypotheses after estimation

[U] **13.5 Accessing coefficients and standard errors**

[U] **20 Estimation and postestimation commands**