Title

stata.com

Bayesian postestimation — Postestimation tools for bayesmh and the bayes prefix

Postestimation commands Remarks and examples Also see

Postestimation commands

The following Bayesian postestimation commands are available after the `bayesmh` command ([BAYES] **bayesmh**) and the `bayes` prefix ([BAYES] **bayes**):

Command	Description
`bayesgraph`	graphical summaries and convergence diagnostics
`bayesstats grubin`	Gelman–Rubin convergence diagnostics
`bayesstats ess`	effective sample sizes and related statistics
`bayesstats ppvalues`	Bayesian predictive p-values (available only after `bayesmh`)
`bayesstats summary`	Bayesian summary statistics for model parameters and their functions
`bayesstats ic`	Bayesian information criteria and Bayes factors
`bayestest model`	hypothesis testing using model posterior probabilities
`bayestest interval`	interval hypothesis testing
`bayespredict`	Bayesian predictions (available only after `bayesmh`)
* `estimates`	cataloging estimation results

* `estimates table` and `estimates stats` are not appropriate with `bayesmh` and `bayes:` estimation results.

Remarks and examples

stata.com

Remarks are presented under the following headings:

After estimation, you can use `bayesgraph` to check convergence of MCMC visually. If you simulated multiple chains, you can use `bayesstats grubin` to compute Gelman–Rubin convergence diagnostics. Once convergence is established, you can use `bayespredict` and `bayesstats ppvalues` to perform model checking after `bayesmh`. Once you are satisfied with the model, you can use `bayesstats summary` to obtain Bayesian summaries such as posterior means and standard deviations of model parameters and functions of model parameters; `bayesstats ess` to compute effective sample sizes and related statistics for model parameters and functions of model parameters; and `bayesstats ic` to compute Bayesian information criteria and Bayes factors for model parameters and their functions. You can use `bayestest model` to test hypotheses by comparing posterior probabilities of models. You can also use `bayestest interval` to test interval hypotheses about parameters and functions of parameters. After `bayesmh`, you can also use `bayespredict` to predict future outcome values.

For an overview example of postestimation commands, see *Overview example* in [BAYES] **Bayesian commands**.

Different ways of specifying model parameters

Many Bayesian postestimation commands such as `bayesstats summary` and `bayesgraph` allow you to specify model parameters for which you want to see the results. To see results for all parameters, simply type a postestimation command without arguments after estimation using `bayesmh` or the `bayes` prefix, for example,

```
. bayesstats summary
```

or you could type

```
. bayesstats summary _all
```

To manually list all model parameters, type

```
. bayesstats summary {param1} {param2} ...
```

or

```
. bayesstats summary {param1 param2} ...
```

The only exception is the `bayesgraph` command when there is more than one model parameter. In that case, `bayesgraph` requires that you either specify `_all` to request all model parameters or specify the model parameters of interest.

You can refer to a single model parameter in the same way you define parameters in, say, the `bayesmh` command. For example, for a parameter with name `param` and no equation name, you can use `{param}`. For a parameter with name `param` and equation name `eqname`, you can use its full name `{eqname:name}`, where the equation name and the parameter name are separated with a colon. With postestimation commands, you can also omit the equation name when referring to the parameter with an equation name.

In the presence of more than one model parameter, you have several ways for referring to multiple parameters at once. If parameters have the same equation name, you can refer to all the parameters with that equation name as follows.

Suppose that you have three parameters with the same equation name `eqname`. Then the specification

```
. bayesstats summary {eqname:param1} {eqname:param2} {eqname:param3}
```

is the same as the specification

```
. bayesstats summary {eqname:}
```

or the specification

```
. bayesstats summary {eqname:param1 param2 param3}
```

The above specification is useful if we want to refer to a subset of parameters with the same equation name. For example, in the above, if we wanted to use only `param1` and `param2`, we could type

```
. bayesstats summary {eqname:param1 param2}
```

There is also a convenient way to refer to the parameters with the same name but different equation names. For example, typing

```
. bayesstats summary {eqname1:param} {eqname2:param}
```

is the same as simply typing

```
. bayesstats summary {param}
```

You can mix and match all the specifications above in one call to a postestimation command. You can also specify expressions of model parameters; see *Specifying functions of model parameters* for details.

Note that if `param` refers to a matrix model parameter, then the results will be provided for all elements of the matrix. For example, if `param` is the name of a 2×2 matrix, then typing

```
. bayesstats summary {param}
```

implies the following:

```
. bayesstats summary {param_1_1} {param_1_2} {param_2_1} {param_2_2}
```

For multilevel models, there are various ways, *reref*, in which you can refer to individual random-effects parameters. Suppose that your model has random intercepts at the `id` level, which are labeled as `{U0[id]}` or `{U0}` for short. To refer to all random intercepts, you can use `{U0}`, `{U0[.]}`, and `{U0[id]}`. To refer to specific random intercepts, you can use `{U0[#]}`, where # refers to the #th element of the random-effects vector, or use `{U0[#.id]}`, where # refers to the #th level of the `id` variable. You can also refer to a subset *numlist* of random intercepts by using `{U0[`*numlist*`]}` or `{U0[(`*numlist*`).id]}`. For nested random effects, for example, `{UU0[id1>id2]}`, you can refer to all random effects as `{UU0}` or `{UU0[.,.]}` and to subsets of random effects as `{UU0[`*numlist*`,`*numlist*`]}` or `{UU0[(`*numlist*`).id1,(`*numlist*`).id2]}`.

Specifying functions of model parameters

You can use Bayesian postestimation commands to obtain results for functions or expressions of model parameters. Each expression must be specified in parentheses. An expression can be any Stata expression, but it may not include matrix model parameters. However, you may include individual elements of matrix model parameters. You may provide labels for your expressions.

For example, we can obtain results for the exponentiated parameter `{param}` as follows:

```
. bayesstats summary (exp({param}))
```

Note that we specified the expression in parentheses.

We can include a label, say, `myexp`, in the above by typing

```
. bayesstats summary (myexp: exp({param}))
```

We can specify multiple expressions by typing

```
. bayesstats summary (myexp: exp({param}) (sd: sqrt({var})))
```

If `param` is a matrix, we can specify expressions, including its elements, but not the matrix itself in the following:

```
. bayesstats summary (exp({param_1_1})) (exp({param_1_2})) ...
```

Storing estimation results after Bayesian estimation

The `bayesmh` command and the `bayes` prefix store various `e()` results such as scalars, macros, and matrices in memory like any other estimation command. Unlike other estimation commands, these commands also save the resulting simulation dataset containing MCMC samples of parameters to disk. Many Bayesian postestimation commands such as `bayesstats summary` and `bayesstats ess` require access to this file. If you do not specify the `saving()` option with `bayesmh` or the `bayes` prefix, the commands save simulation results in a temporary Stata dataset. This file is being

replaced with the new simulation results each time bayesmh or the bayes prefix is run. To save your simulation results, you must specify the saving() option with bayesmh or the bayes prefix, in which case your simulation results are saved to the specified file in the specified location and will not be overridden by the next call to these commands.

You can specify the saving() option during estimation by typing

```
. bayesmh ..., likelihood() prior() ... saving()
```

or

```
. bayes, saving(): ...
```

or on replay by typing

```
. bayesmh, saving()
```

or

```
. bayes, saving()
```

As you can with other estimation commands, you can use estimates store to store Bayesian estimation results in memory and estimates save to save them to disk, but you must first use the saving() option with bayesmh or the bayes prefix to save simulation data in a permanent dataset. For example, type

```
. bayesmh ..., likelihood() prior() ... saving(bmh_simdata)
. estimates store model1
```

or, after bayesmh estimation, type

```
. bayesmh, saving(bmh_simdata)
. estimates store model1
```

Once you create a permanent dataset, it is your responsibility to erase it after it is no longer needed. estimates drop and estimates clear will drop estimation results only from memory; they will not erase the simulation files you saved.

```
. estimates drop model1
. erase bmh_simdata.dta
```

See [R] **estimates** for more information about commands managing estimation results. estimates table and estimates stats are not appropriate after bayesmh and the bayes prefix.

Different ways of specifying predictions and their functions

After bayesmh, you can use the bayespredict command to simulate outcome variables, residuals, and other test quantities; see [BAYES] **bayespredict**. Bayesian postestimation commands bayesgraph, bayesstats summary, bayesstats ppvalues, bayesstats ess, and bayestest interval can then be used to obtain graphs, posterior summaries, and so on for these prediction quantities.

In this section, we describe various specifications of prediction results with Bayesian postestimation commands mentioned above. We use bayesstats summary in our examples, but the same specifications may be used with other postestimation commands, except that bayestest interval allows only specifications containing individual observations.

Suppose that we use the bayesmh command to fit a model with two outcome variables.

```
. bayesmh y1 y2 = x1 x2, ... saving(mcmcfile)
```

We then use bayespredict to simulate samples for these two outcome variables and save them in a prediction dataset, predfile.dta.

```
. bayespredict {_ysim1} {_ysim2}, saving(predfile)
```

To access prediction results, all postestimation commands must specify the prediction dataset in the using specification. In fact, this is all postestimation commands need to produce results for the prediction quantities. (Technically, the auxiliary estimation file generated by bayespredict, for example, predfile.ster, must also exist.) That is, they do not rely on the estimation results or the simulation data from bayesmh.

When the prediction dataset contains simulated outcomes, in addition to accessing these outcomes (for instance, {_ysim1} and {_ysim2} in our example), postestimation commands may also access the residuals ({_resid1} and {_resid2}), expected values ({_mu1} and {_mu2}), and Stata expressions of simulated outcomes, residuals, and expected values. You can also call Mata functions within command specifications to compute functions of simulated outcomes, residuals, and expected values.

Let's calculate posterior summaries for all observations of the first outcome and for all residuals of the second outcome.

```
. bayesstats summary {_ysim1} {_resid2} using predfile
```

You can refer to a subset of predicted observations, say, from 1 to 10 for the observations and from 1 to 5 for the residuals.

```
. bayesstats summary {_ysim1[1/10]} {_resid2[1/5]} using predfile
```

You can compute expressions of individual simulated outcome observations and their residuals.

```
. bayesstats summary (exp({_ysim1[1]})) ({_resid2[1]}^2) using predfile
```

You can test whether the residual for the first observation of the second outcome variable is greater than zero by using bayestest interval to calculate the corresponding posterior probability.

```
. bayestest interval {_resid2[1]} using predfile, lower(0)
```

As we mentioned earlier, you can use Mata functions of predicted outcomes and residuals. These functions operate across observations. For example, to summarize the mean of the first simulated outcome and the variance of the second simulated outcome, type

```
. bayesstats summary (@mean({_ysim1})) (@variance({_ysim2})) using predfile
```

Instead of using the default labels for the computed quantities, you can specify your own. Below, we use mean and var to label the corresponding predictions.

```
. bayesstats summary (mean:@mean({_ysim1})) (var:@variance({_ysim2})) using predfile
```

You cannot specify Mata functions with bayestest interval, and, unlike bayespredict, you cannot specify Stata programs within the postestimation commands.

If you need to access individual values of the predicted quantity computed using a Mata function or specify an expression of this quantity, you need to compute and save this quantity with bayespredict.

Suppose that you wish to compute the sum of the two outcome variances. You simulate these variances by using bayespredict first.

```
. bayespredict (prvar1:@variance({_ysim1})) (prvar2:@variance({_ysim2})),  ///
        saving(predfile)
```

In the above, we labeled the computed variances as prvar1 and prvar2.

Then, you can call `bayesstats summary` to compute the sum of the predicted quantities.

```
. bayesstats summary ({prvar1} + {prvar2}) using predfile
```

Or you can obtain summaries of each predicted quantity.

```
. bayesstats summary {prvar1} {prvar2} using predfile
```

You can combine various specifications in one call to the postestimation command. For example, let's save the following prediction quantities with `bayespredict`.

```
. bayespredict {_ysim1} {_ysim2} (mean1:@mean({_ysim1}))  ///
        (var2:@variance({_ysim2})), saving(predfile)
```

You can specify multiple prediction quantities in one call to `bayesstats summary` or other postestimation commands.

```
. bayesstats summary ({_ysim1}) ({_resid[1/5]}) ({mean1})    ///
        ({var2}) (mean2:@mean({_ysim2})) using predfile
```

Also see

[BAYES] **bayes** — Bayesian regression models using the bayes prefix

[BAYES] **bayesmh** — Bayesian models using Metropolis–Hastings algorithm

[BAYES] **bayesmh evaluators** — User-defined evaluators with bayesmh

[BAYES] **Bayesian commands** — Introduction to commands for Bayesian analysis

[BAYES] **Intro** — Introduction to Bayesian analysis

[BAYES] **Glossary**

[U] 20 Estimation and postestimation commands