bayesmh evaluators — User-defined evaluators with bayesmh

Description Syntax Options Remarks and examples
Stored results Reference Also see

Description

bayesmh provides two options, evaluator() and llevaluator(), that facilitate user-defined evaluators for fitting general Bayesian regression models. bayesmh, evaluator() accommodates log-posterior evaluators. bayesmh, llevaluator() accommodates log-likelihood evaluators, which are combined with built-in prior distributions to form the desired posterior density. For a catalog of built-in likelihood models and prior distributions, see [BAYES] **bayesmh**.

Syntax

Single-equation models

User-defined log-posterior evaluator

bayesmh *depvar* [*indepvars*] [*if*] [*in*] [*weight*] , evaluator(*evalspec*) [*options*]

User-defined log-likelihood evaluator

bayesmh *depvar* [*indepvars*] [*if*] [*in*] [*weight*] , llevaluator(*evalspec*)
prior(*priorspec*) [*options*]

Multiple-equations models

User-defined log-posterior evaluator

bayesmh (*eqspecp*) [(*eqspecp*) [...]] [*if*] [*in*] [*weight*] , evaluator(*evalspec*)
[*options*]

User-defined log-likelihood evaluator

bayesmh (*eqspecll*) [(*eqspecll*) [...]] [*if*] [*in*] [*weight*] , prior(*priorspec*)
[*options*]

The syntax of *eqspecp* is

varspec [, noconstant]

The syntax of *eqspecll* for built-in likelihood models is

varspec , likelihood(*modelspec*) [noconstant]

The syntax of *eqspecll* for user-defined log-likelihood evaluators is

varspec , llevaluator(*evalspec*) [noconstant]

The syntax of *varspec* is one of the following:

for single outcome

[*eqname* :] *depvar* [*indepvars*]

for multiple outcomes with common regressors

depvars = [*indepvars*]

for multiple outcomes with outcome-specific regressors

([*eqname1* :] *depvar1* [*indepvars1*]) ([*eqname2* :] *depvar2* [*indepvars2*]) [. . .]

The syntax of *evalspec* is

progname , parameters(*paramlist*) [extravars(*varlist*) passthruopts(*string*)
reparameters(*reparamlist*) predict]

progname is the name of a Stata program that you write to evaluate the log-posterior density or the log-likelihood function (see User-defined evaluators). *paramlist* is a list of model parameters:

paramdef [*paramdef* [. . .]]

reparamlist is a list of random-effects model parameters:

reparamdef [*reparamdef* [. . .]]

The syntax of *paramdef* is

{ [*eqname* :] *param* [*param* [. . .]] [, matrix] }

The parameter label *eqname* and parameter names *param* are valid Stata names. Model parameters are either scalars such as {var}, {mean}, and {shape:alpha} or matrices such as {Sigma, matrix} and {Scale:V, matrix}. For scalar parameters, you can use {param=#} in the above to specify an initial value. For example, you can specify {var=1}, {mean=1.267}, or {shape:alpha=3}. You can specify the multiple parameters with same equation as {eq:p1 p2 p3} or {eq: S1 S2, matrix}. Also see *Declaring model parameters* in [BAYES] **bayesmh**.

The syntax of *reparamdef* is

{*rename* [*levelspec*] }

rename is a Stata name that starts with a capital letter, and *levelspec* describes the level specification; see Random effects in [BAYES] **bayesmh**.

options	Description
* evaluator(*evalspec*)	specify log-posterior evaluator; may not be combined with llevaluator() and prior()
* llevaluator(*evalspec*)	specify log-likelihood evaluator; requires prior() and may not be combined with evaluator()
* prior(*priorspec*)	specify prior for model parameters; required with log-likelihood evaluator and may be repeated
likelihood(*modelspec*)	specify distribution for the likelihood model; allowed within an equation of a multiple-equations model only
noconstant	suppress constant term; not allowed with ordered models specified in likelihood() with multiple-equations models
scalarlnden	specify that the evaluator return a scalar log-density value
bayesmhopts	any options of [BAYES] **bayesmh** except likelihood() and prior()

* Option evaluator() is required for log-posterior evaluators, and options llevaluator() and prior() are required for log-likelihood evaluators. With log-likelihood evaluators, prior() must be specified for all model parameters and may be repeated.

indepvars may contain factor variables; see [U] **11.4.3 Factor variables**.

Only fweights are allowed; see [U] **11.1.6 weight**.

Options

evaluator(*evalspec*) specifies the name and the attributes of the log-posterior evaluator; see *User-defined evaluators* for details. This option may not be combined with llevaluator() or likelihood().

llevaluator(*evalspec*) specifies the name and the attributes of the log-likelihood evaluator; see *User-defined evaluators* for details. This option may not be combined with evaluator() or likelihood() and requires the prior() option.

prior(*priorspec*); see [BAYES] **bayesmh**.

likelihood(*modelspec*); see [BAYES] **bayesmh**. This option is allowed within an equation of a multiple-equations model only.

noconstant; see [BAYES] **bayesmh**.

scalarlnden specifies that the evaluator return a scalar log-density value. Specifically, when this option is specified, likelihood evaluators must return the total log-likelihood value over the estimation sample, and posterior evaluators must return the log-posterior value. Without this option, both likelihood and posterior evaluators are expected to return the observation-specific likelihood values; in addition, posterior evaluators are expected to return a scalar log-prior value. bayesmh then automatically combines the provided information to form the final log-posterior value. This option may not be combined with likelihood().

bayesmhopts specify any *options* of [BAYES] **bayesmh**, except likelihood() and prior().

Remarks and examples

Remarks are presented under the following headings:

User-defined evaluators
Simple linear regression model
Simple linear regression model with scalar evaluators
Logistic regression model
Multivariate normal regression model
Cox proportional hazards regression
Random-intercept linear regression model
Evaluators with predictions
Global macros

User-defined evaluators

If your likelihood model or prior distributions are particularly complex and cannot be represented by one of the predefined sets of distributions or by substitutable expressions provided with `bayesmh`, you can program these functions by writing your own evaluator program.

Evaluator programs can be used for programming the full posterior density by specifying the `evaluator()` option or only the likelihood portion of your Bayesian model by specifying the `llevaluator()` option. For likelihood evaluators, `prior()` option(s) must be specified for all model parameters. Without the `scalarlnden` option, your program is expected to calculate and return individual log-likelihood values, one for each observation in the estimation sample. The posterior evaluator must also calculate and return the scalar log-prior value. When the `scalarlnden` option is specified, your program is expected to calculate and return a total (overall) log-likelihood density value with likelihood evaluators and a log-posterior density value with posterior evaluators.

It is allowed for the return values to match the log density up to an additive constant, in which case, however, some of the reported statistics such as DIC and log marginal-likelihood may not be applicable.

Your evaluator program *progname* must be a Stata program; see **[U] 18 Programming Stata**. The program must follow one of the styles below.

Program for log-posterior evaluators:

```
program  progname
    args lnfj lnprior xb1 [xb2 ...] [ modelparams] [ reparamlist]
    ... computations ...
    replace ‘lnfj’ = ... if $MH_touse
    ... computations ...
    scalar ‘lnprior’ = ...
end
```

Program for log-likelihood evaluators:

```
program  progname
    args lnfj xb1 [xb2 ...] [ modelparams] [ reparamlist]
    ... computations ...
    replace ‘lnfj’ = ... if $MH_touse
end
```

`lnfj` contains the name of a temporary variable to be filled in with observation-specific log-likelihood values.

`lnprior` contains the name of a temporary scalar to be filled in with the log-prior value.

xb# contains the name of a temporary variable where the linear predictor of the #th equation is stored.

modelparams is a list of names of scalars or matrices to contain the values of model parameters specified in suboption parameters() of evaluator() or llevaluator(). For matrix parameters, the specified names will contain the names of temporary matrices where the current values are stored. For scalar parameters, these are the names of temporary scalars containing current values. The order in which names are listed should correspond to the order in which model parameters are specified in parameters().

reparamlist is a list of names of temporary variables to contain the values of random-effects parameters specified with option reparameters(). These are the random-effects parameters you may want to have an easy access to in the evaluator program. The order of the names matches the order of the random-effects parameters specified in reparameters().

When the scalarlnden option is specified, the program syntax for both posterior and likelihood evaluators is

```
program progname
    args lnden xb1 [xb2 ...] [ modelparams] [ reparamlist]
    ... computations ...
    scalar ‘lnden’ = ...
end
```

lnden contains the name of a temporary scalar to be filled in with an overall log-posterior or log-likelihood value.

Also see *Global macros* for a list of global macros available in the evaluator program.

After you write an evaluator program, you specify its name in the option evaluator() for log-posterior evaluators,

```
. bayesmh ..., evaluator(progname, evalopts)
```

or option llevaluator() for log-likelihood evaluators,

```
. bayesmh ..., llevaluator(progname, evalopts)
```

Evaluator options *evalopts* include parameters(), extravars(), passthruopts(), reparameters(), and predict.

parameters(*paramlist*) specifies model parameters. Model parameters can be scalars or matrices. Each parameter must be specified in curly braces {}. Multiple parameters with the same equation names may be specified within one set of {}.

For example,

```
parameters({mu} {var:sig2} {S,matrix} {cov:Sigma, matrix} {prob:p1 p2})
```

specifies a scalar parameter with name mu without an equation label, a scalar parameter with name sig2 and label var, a matrix parameter with name S, a matrix parameter with name Sigma and label cov, and two scalar parameters {prob:p1} and {prob:p2}.

extravars(*varlist*) specifies any variables in addition to dependent and independent variables that you may need in your evaluator program. Examples of such variables are offset variables, exposure variables for count-data models, and failure or censoring indicators for survival-time models. See *Cox proportional hazards regression* for an example.

passthruopts(*string*) specifies a list of options you may want to pass to your evaluator program. For example, these options may contain fixed values of model parameters and hyperparameters. See *Multivariate normal regression model* for an example.

`reparameters(`*reparamlist*`)` specifies random-effects model parameters. This option is useful when you need to perform computations that require direct access to the random-effects parameters in the evaluator. Otherwise, you may simply use the linear predictor `xb#`'s, which automatically include the random effects.

For example,

```
reparameters({U1[id]} {U2[id2>id1]} {W[_n]})
```

specifies a random-effects parameter `U1` with the group variable `id`, a random-effects parameter `U2` with the group variable `id2` nested within the group `id1`, and a latent variable `W`.

`predict` specifies that the evaluator include the code to generate random samples for the outcome from its likelihood model; see *Prior and posterior predictive distributions* in [BAYES] **bayespredict**. If this option is not specified for the evaluator in your `bayesmh` command, calling `bayespredict` afterward to obtain predictions for the corresponding outcome will result in an error. With multiple outcomes and evaluators, option `predict` is evaluator specific: you may implement predictions for only some of the outcomes. For examples, see *Evaluators with predictions* in [BAYES] **bayespredict** and *Evaluators with predictions*.

`bayesmh` automatically creates parameters for regression coefficients: `{`*depname*`:`*varname*`}` for every *varname* in *indepvars*, and a constant parameter `{`*depname*`:_cons}` unless `noconstant` is specified. These parameters are used to form linear predictors used in the evaluator program. If you need to access values of the parameters in the evaluator, you can use `$MH_b`; see the log-posterior evaluator in *Cox proportional hazards regression* for an example. With multiple dependent variables, regression coefficients are defined for each dependent variable.

Simple linear regression model

Suppose that we want to fit a Bayesian normal regression where we program the posterior distribution ourselves. The `normaljeffreys` program below computes the log-posterior density for the normal linear regression with flat priors for the coefficients and the Jeffreys prior for the variance parameter.

```
. program normaljeffreys
  1.        version 19.5      // (or version 19 if you do not have StataNow)
  2.        args lnfj lnprior xb var
  3.        /* compute log likelihood */
   .        tempname sd
  4.        scalar `sd' = sqrt(`var')
  5.        quietly replace `lnfj' = lnnormalden($MH_y,`xb',`sd') if $MH_touse
  6.        /* compute log prior */
  7.        scalar `lnprior' = -2*ln(`sd')
  8. end
```

The program accepts four parameters: the temporary variable name `` `lnfj' `` to contain the observation-specific log-likelihood values, the temporary name `` `lnprior' `` of a scalar to contain the log-prior value, the temporary name `` `xb' `` of the variable that contains the linear predictor, and the temporary name `` `var' `` of a scalar that contains the value of the variance parameter.

The first part of the program calculates the observation-specific log likelihood of the normal regression. The second part of the program calculates the log of prior distributions of the parameters. Because the coefficients have flat prior distributions with densities of 1, their log is 0 and does not contribute to the overall prior. The only contribution is from the Jeffreys prior $\ln(1/\sigma^2) = -2\ln(\sigma)$ for the variance σ^2. As the final step, `bayesmh` automatically computes the value of the posterior density as the sum of the total (overall) log likelihood and the log of the prior.

The substantial portion of this program is the computation of the log likelihood. The global macro `$MH_y` contains the name of the dependent variable, and `$MH_touse` contains a temporary marker variable identifying observations to be used in the computations.

We used the built-in function `lnnormalden()` to compute observation-specific log likelihood. The temporary variable `‘lnfj’` is created by `bayesmh`, and you need to replace only its values. (If you create a temporary variable yourself for intermediate calculations, remember to create it of type `double` to ensure the highest precision of the results.) It is also important to perform computations using only the relevant subset of observations as identified by the marker variable stored in `$MH_touse`. This variable contains the value of 1 for observations to be used in the computations and 0 for the remaining observations. Missing values in used variables affect this variable, as do the qualifiers `if` and `in` of the `bayesmh` command.

We can now specify the `normaljeffreys` evaluator in the `evaluator()` option of `bayesmh`. In addition to the regression coefficients, we have one extra parameter, the variance of the normal distribution, which we must specify in the `parameters()` suboption of `evaluator()`.

We use `auto.dta` to illustrate the command. We specify a simple regression of `mpg` on rescaled `weight`.

```
. use https://www.stata-press.com/data/r19/auto
(1978 automobile data)
. quietly replace weight = weight/100
. set seed 14
. bayesmh mpg weight, evaluator(normaljeffreys, parameters({var}))
Burn-in ...
note: invalid initial state.
Simulation ...
Model summary
------------------------------------------------------------------------------
Posterior:
  mpg ~ normaljeffreys(xb_mpg,{var})
------------------------------------------------------------------------------

Bayesian regression                              MCMC iterations  =     12,500
Random-walk Metropolis-Hastings sampling         Burn-in          =      2,500
                                                 MCMC sample size =     10,000
                                                 Number of obs    =         74
                                                 Acceptance rate  =      .1433
                                                 Efficiency:  min =     .06246
                                                              avg =     .06669
Log marginal-likelihood =   -198.247                          max =     .07091

------------------------------------------------------------------------------
             |                                                Equal-tailed
             |      Mean   Std. dev.     MCSE     Median  [95% cred. interval]
-------------+----------------------------------------------------------------
mpg          |
      weight | -.6052218    .053604   .002075  -.6062666  -.7121237  -.4992178
       _cons |  39.56782   1.658124   .066344   39.54211   36.35645   42.89876
-------------+----------------------------------------------------------------
         var |  12.19046   2.008871   .075442   12.03002   8.831172   17.07787
------------------------------------------------------------------------------
```

The output of `bayesmh` with user-defined evaluators is the same as the output of `bayesmh` with built-in distributions, except the title and the model summary. The generic title `Bayesian regression` is used for all evaluators, but you can change it by specifying the `title()` option. The model summary provides the name of the posterior evaluator.

Following the command line, there is a note about invalid initial state. For program evaluators, bayesmh initializes all parameters with zeros, except for positive parameters used in prior specifications, which are initialized with ones. This may not be sensible for all parameters, such as the variance parameter in our example. We may consider using, for example, OLS estimates as initial values of the parameters.

We now specify initial values in the initial() option.

```
. set seed 14

. bayesmh mpg weight, evaluator(normaljeffreys, parameters({var}))
> initial({mpg:weight} -0.6 {mpg:_cons} 39 {var} 11.83)

Burn-in ...
Simulation ...

Model summary
------------------------------------------------------------------------------
Posterior:
  mpg ~ normaljeffreys(xb_mpg,{var})
------------------------------------------------------------------------------

Bayesian regression                              MCMC iterations  =     12,500
Random-walk Metropolis-Hastings sampling         Burn-in          =      2,500
                                                 MCMC sample size =     10,000
                                                 Number of obs    =         74
                                                 Acceptance rate  =      .1668
                                                 Efficiency:  min =     .04114
                                                              avg =     .04811
Log marginal-likelihood = -198.14302                          max =     .05938

------------------------------------------------------------------------------
             |                                                Equal-tailed
             |      Mean   Std. dev.     MCSE     Median  [95% cred. interval]
-------------+----------------------------------------------------------------
mpg          |
      weight | -.6025616   .0540995   .002667  -.6038729  -.7115221  -.5005915
       _cons |  39.50491   1.677906   .080156   39.45537    36.2433   43.14319
-------------+----------------------------------------------------------------
         var |  12.26586   2.117858   .086915   12.05298   8.827655   17.10703
------------------------------------------------------------------------------
```

We can compare our results with results from bayesmh, which uses a built-in normal likelihood and flat and Jeffreys priors. To match the results, we must use the same initial values, because bayesmh has a different initialization logic for built-in distributions.

```
. set seed 14

. bayesmh mpg weight, likelihood(normal({var}))
> prior({mpg:}, flat) prior({var}, jeffreys)
> initial({mpg:weight} -0.6 {mpg:_cons} 39 {var} 11.83)

Burn-in ...
Simulation ...

Model summary
------------------------------------------------------------------------------
Likelihood:
  mpg ~ normal(xb_mpg,{var})

Priors:
  {mpg:weight _cons} ~ 1 (flat)                                            (1)
               {var} ~ jeffreys
------------------------------------------------------------------------------
(1) Parameters are elements of the linear form xb_mpg.

Bayesian normal regression                       MCMC iterations  =     12,500
Random-walk Metropolis-Hastings sampling         Burn-in          =      2,500
                                                 MCMC sample size =     10,000
                                                 Number of obs    =         74
                                                 Acceptance rate  =      .1668
                                                 Efficiency:  min =     .04114
                                                              avg =     .04811
Log marginal-likelihood = -198.14302                          max =     .05938

------------------------------------------------------------------------------
             |                                                Equal-tailed
             |      Mean   Std. dev.     MCSE     Median  [95% cred. interval]
-------------+----------------------------------------------------------------
mpg          |
      weight | -.6025616   .0540995   .002667  -.6038729  -.7115221  -.5005915
       _cons |  39.50491   1.677906   .080156   39.45537    36.2433   43.14319
-------------+----------------------------------------------------------------
         var |  12.26586   2.117858   .086915   12.05298   8.827655   17.10703
------------------------------------------------------------------------------
```

If your Bayesian model uses prior distributions that are supported by bayesmh but the likelihood model is not supported, you can write only the likelihood evaluator and use built-in prior distributions.

For example, we can place the portion of the normaljeffreys program that computes the log likelihood in a separate program and call it normalreg.

```
. program normalreg
  1.          version 19.5      // (or version 19 if you do not have StataNow)
  2.          args lnfj xb var
  3.          /* compute log likelihood */
  4.          tempname sd
  5.          scalar `sd' = sqrt(`var')
  6.          quietly replace `lnfj' = lnnormalden($MH_y,`xb',`sd') if $MH_touse
  7. end
```

We can now specify this program in the llevaluator() option and use prior() options to specify built-in flat priors for the coefficients and the Jeffreys prior for the variance.

```
. set seed 14

. bayesmh mpg weight, llevaluator(normalreg, parameters({var}))
> prior({mpg:}, flat) prior({var}, jeffreys)
> initial({mpg:weight} -0.6 {mpg:_cons} 39 {var} 11.83)

Burn-in ...
Simulation ...

Model summary
------------------------------------------------------------------------------
Likelihood:
  mpg ~ normalreg(xb_mpg,{var})

Priors:
  {mpg:weight _cons} ~ 1 (flat)                                            (1)
               {var} ~ jeffreys
------------------------------------------------------------------------------
(1) Parameters are elements of the linear form xb_mpg.

Bayesian regression                              MCMC iterations  =     12,500
Random-walk Metropolis-Hastings sampling         Burn-in          =      2,500
                                                 MCMC sample size =     10,000
                                                 Number of obs    =         74
                                                 Acceptance rate  =      .1668
                                                 Efficiency:  min =     .04114
                                                              avg =     .04811
Log marginal-likelihood = -198.14302                          max =     .05938

------------------------------------------------------------------------------
             |                                                Equal-tailed
             |      Mean   Std. dev.     MCSE     Median  [95% cred. interval]
-------------+----------------------------------------------------------------
mpg          |
      weight | -.6025616   .0540995   .002667  -.6038729  -.7115221  -.5005915
       _cons |  39.50491   1.677906   .080156   39.45537    36.2433   43.14319
-------------+----------------------------------------------------------------
         var |  12.26586   2.117858   .086915   12.05298   8.827655   17.10703
------------------------------------------------------------------------------
```

We obtain the same results as earlier.

Simple linear regression model with scalar evaluators

Here we show a scalar version of the `normaljeffreys` program that computes the total log likelihood, adds it to the log prior, and returns the final log posterior as a scalar value.

```
. program normaljeffreys2
  1.          version 19.5      // (or version 19 if you do not have StataNow)
  2.          args lnp xb var
  3.          /* compute log likelihood */
   .          tempname sd
  4.          scalar ‘sd’ = sqrt(‘var’)
  5.          tempvar lnfj
  6.          quietly generate double ‘lnfj’=lnnormalden($MH_y,‘xb’,‘sd’) if
> $MH_touse
  7.          quietly summarize ‘lnfj’, meanonly
  8.          if r(N) < $MH_n {
  9.                  scalar ‘lnp’ = .
 10.                  exit
 11.          }
 12.          tempname lnf
 13.          scalar ‘lnf’ = r(sum)
 14.          /* compute log prior */
   .          tempname lnprior
 15.          scalar ‘lnprior’ = -2*ln(‘sd’)
 16.          /* compute log posterior */
   .          scalar ‘lnp’ = ‘lnf’ + ‘lnprior’
 17. end
```

Here we created the temporary variable `‘lnfj’` ourselves to contain the observation-specific log-likelihood values. And we used `summarize` to obtain the total value. After we compute the log-likelihood value, we should verify that the number of nonmissing observation-specific contributions to the log likelihood equals `$MH_n`. If it does not, the log-posterior value (or log-likelihood value in a log-likelihood evaluator) must be set to missing. (`$MH_n` contains the total number of observations in the sample identified by the `$MH_touse` variable.) Unlike in our previous example programs, here we compute the log-posterior value ourselves.

We refit the first model from the previous section but now using the `normaljeffreys2` evaluator. Because the evaluator now returns the scalar log posterior, we also need to add the `scalarlnden` option to the `bayesmh` specification.

```
. set seed 14

. bayesmh mpg weight, evaluator(normaljeffreys2, parameters({var})) scalarlnden

Burn-in ...
note: invalid initial state.
Simulation ...

Model summary
------------------------------------------------------------------------------
Posterior:
  mpg ~ normaljeffreys2(xb_mpg,{var})
------------------------------------------------------------------------------

Bayesian regression                              MCMC iterations  =     12,500
Random-walk Metropolis-Hastings sampling         Burn-in          =      2,500
                                                 MCMC sample size =     10,000
                                                 Number of obs    =         74
                                                 Acceptance rate  =      .1433
                                                 Efficiency:  min =     .06246
                                                              avg =     .06669
Log marginal-likelihood =   -198.247                          max =     .07091

------------------------------------------------------------------------------
             |                                                Equal-tailed
             |      Mean   Std. dev.     MCSE     Median  [95% cred. interval]
-------------+----------------------------------------------------------------
mpg          |
      weight | -.6052218    .053604   .002075  -.6062666  -.7121237  -.4992178
       _cons |  39.56782   1.658124   .066344   39.54211   36.35645   42.89876
-------------+----------------------------------------------------------------
         var |  12.19046   2.008871   .075442   12.03002   8.831172   17.07787
------------------------------------------------------------------------------
```

For this simple linear regression model, the `normaljeffreys` and `normaljeffreys2` evaluators produce the same results.

Next we show a scalar version of the `normalreg` likelihood evaluator. The evaluator computes and returns the total log likelihood over the estimation sample.

```
. program normalreg2
  1.         version 19.5      // (or version 19 if you do not have StataNow)
  2.         args lnf xb var
  3.         /* compute log likelihood */
   .         tempname sd
  4.         scalar `sd' = sqrt(`var')
  5.         tempvar lnfj
  6.         quietly generate double `lnfj' = lnnormalden($MH_y,`xb',`sd') if
> $MH_touse
  7.         quietly summarize `lnfj', meanonly
  8.         if r(N) < $MH_n {
  9.                 scalar `lnf' = .
 10.                 exit
 11.         }
 12.         scalar `lnf' = r(sum)
 13. end
```

We refit the last model from the previous section but this time using the normalreg2 likelihood evaluator and also specifying the scalarlnden option.

```
. set seed 14
. bayesmh mpg weight, llevaluator(normalreg2, parameters({var})) scalarlnden
> prior({mpg:}, flat) prior({var}, jeffreys)
> initial({mpg:weight} -0.6 {mpg:_cons} 39 {var} 11.83)
Burn-in ...
Simulation ...
Model summary
------------------------------------------------------------------------------
Likelihood:
  mpg ~ normalreg2(xb_mpg,{var})
Priors:
  {mpg:weight _cons} ~ 1 (flat)                                            (1)
               {var} ~ jeffreys
------------------------------------------------------------------------------
(1) Parameters are elements of the linear form xb_mpg.
Bayesian regression                              MCMC iterations  =     12,500
Random-walk Metropolis-Hastings sampling         Burn-in          =      2,500
                                                 MCMC sample size =     10,000
                                                 Number of obs    =         74
                                                 Acceptance rate  =      .1668
                                                 Efficiency:  min =     .04114
                                                              avg =     .04811
Log marginal-likelihood = -198.14302                          max =     .05938
```

	Mean	Std. dev.	MCSE	Median	Equal-tailed [95% cred. interval]	
mpg						
weight	-.6025616	.0540995	.002667	-.6038729	-.7115221	-.5005915
_cons	39.50491	1.677906	.080156	39.45537	36.2433	43.14319
var	12.26586	2.117858	.086915	12.05298	8.827655	17.10703

Logistic regression model

Some models, such as logistic regression, do not have additional parameters except regression coefficients. Here we show how to write an evaluator program for fitting a Bayesian logistic regression model.

We start by creating a program for computing the log likelihood.

```
. program logitll
  1.         version 19.5      // (or version 19 if you do not have StataNow)
  2.         args lnfj xb
  3.         quietly replace `lnfj' = ln(invlogit( `xb'))
>                                       if $MH_y == 1 & $MH_touse
  4.         quietly replace `lnfj'  = ln(invlogit(-`xb'))
>                                       if $MH_y == 0 & $MH_touse
  5. end
```

The structure of our log-likelihood evaluator is similar to the one described in *Simple linear regression model*, except we have no extra parameters.

We continue with auto.dta and regress foreign on mpg. For simplicity, we assume a flat prior for the coefficients and use bayesmh, llevaluator() to fit this model.

```
. use https://www.stata-press.com/data/r19/auto, clear
(1978 automobile data)

. set seed 14

. bayesmh foreign mpg, llevaluator(logitll) prior({foreign:}, flat)

Burn-in ...
Simulation ...

Model summary
------------------------------------------------------------------------------
Likelihood:
  foreign ~ logitll(xb_foreign)

Prior:
  {foreign:mpg _cons} ~ 1 (flat)                                           (1)
------------------------------------------------------------------------------
(1) Parameters are elements of the linear form xb_foreign.

Bayesian regression                              MCMC iterations  =     12,500
Random-walk Metropolis-Hastings sampling         Burn-in          =      2,500
                                                 MCMC sample size =     10,000
                                                 Number of obs    =         74
                                                 Acceptance rate  =      .2216
                                                 Efficiency:  min =     .09293
                                                              avg =     .09989
Log marginal-likelihood = -41.626029                          max =      .1068

------------------------------------------------------------------------------
             |                                                Equal-tailed
     foreign |      Mean   Std. dev.     MCSE     Median  [95% cred. interval]
-------------+----------------------------------------------------------------
         mpg |    .16716   .0545771    .00167   .1644019   .0669937   .2790017
       _cons | -4.560636   1.261675   .041387  -4.503921   -7.10785  -2.207665
------------------------------------------------------------------------------
```

The results from the evaluator version match the results from bayesmh with a built-in logistic model.

```
. set seed 14
. bayesmh foreign mpg, likelihood(logit) prior({foreign:}, flat)
> initial({foreign:} 0)
Burn-in ...
Simulation ...
Model summary
------------------------------------------------------------------------------
Likelihood:
  foreign ~ logit(xb_foreign)
Prior:
  {foreign:mpg _cons} ~ 1 (flat)                                          (1)
------------------------------------------------------------------------------
(1) Parameters are elements of the linear form xb_foreign.
Bayesian logistic regression                     MCMC iterations  =     12,500
Random-walk Metropolis-Hastings sampling         Burn-in          =      2,500
                                                 MCMC sample size =     10,000
                                                 Number of obs    =         74
                                                 Acceptance rate  =      .2216
                                                 Efficiency:  min =     .09293
                                                              avg =     .09989
Log marginal-likelihood = -41.626029                          max =      .1068

------------------------------------------------------------------------------
             |                                                Equal-tailed
     foreign |      Mean   Std. dev.     MCSE     Median  [95% cred. interval]
-------------+----------------------------------------------------------------
         mpg |    .16716   .0545771     .00167   .1644019   .0669937   .2790017
       _cons | -4.560636   1.261675    .041387  -4.503921   -7.10785  -2.207665
------------------------------------------------------------------------------
```

Because we assumed a flat prior with the density of 1, the log prior is 0, so the log-posterior evaluator for this model is the same as the log-likelihood evaluator.

```
. program logitposter
  1.         version 19.5      // (or version 19 if you do not have StataNow)
  2.         args lnfj lnprior xb
  3.         quietly replace ‘lnfj’ = ln(invlogit( ‘xb’))
>                                        if $MH_y == 1 & $MH_touse
  4.         quietly replace ‘lnfj’  = ln(invlogit(-‘xb’))
>                                        if $MH_y == 0 & $MH_touse
  5.         scalar ‘lnprior’ = 0
  6. end

. set seed 14

. bayesmh foreign mpg, evaluator(logitposter)

Burn-in ...
Simulation ...

Model summary
------------------------------------------------------------------------------
Posterior:
  foreign ~ logitposter(xb_foreign)
------------------------------------------------------------------------------

Bayesian regression                              MCMC iterations  =     12,500
Random-walk Metropolis-Hastings sampling         Burn-in          =      2,500
                                                 MCMC sample size =     10,000
                                                 Number of obs    =         74
                                                 Acceptance rate  =      .2216
                                                 Efficiency:  min =     .09293
                                                              avg =     .09989
Log marginal-likelihood = -41.626029                          max =      .1068

------------------------------------------------------------------------------
             |                                                Equal-tailed
     foreign |      Mean   Std. dev.     MCSE     Median  [95% cred. interval]
-------------+----------------------------------------------------------------
         mpg |    .16716   .0545771    .00167   .1644019   .0669937   .2790017
       _cons | -4.560636   1.261675   .041387  -4.503921   -7.10785  -2.207665
------------------------------------------------------------------------------
```

Multivariate normal regression model

Here we demonstrate how to write an evaluator program for a multivariate response. We consider a bivariate normal regression, and we again start with a log-likelihood evaluator. In this example, we also use Mata to speed up our computations.

```
. program mvnregll
  1.         version 19.5      // (or version 19 if you do not have StataNow)
  2.         args lnfj xb1 xb2
  3.         tempvar diff1 diff2 touseid
  4.         quietly generate double ‘diff1’ = $MH_y1 - ‘xb1’ if $MH_touse
  5.         quietly generate double ‘diff2’ = $MH_y2 - ‘xb2’ if $MH_touse
  6.         local d $MH_yn
  7.         local n $MH_n
  8.         quietly generate ‘touseid’ = $MH_touse * _n
  9.         mata: st_store(st_data(.,"‘touseid’"), "‘lnfj’", ///
>                  mvnll_mata(‘d’,‘n’,"‘diff1’","‘diff2’"))
 10. end
```

```
. mata:
------------------------------------------------- mata (type end to exit) -----
: real vector mvnll_mata(real scalar d, n, string scalar sdiff1, sdiff2)
> {
>         real vector lnfj, vcross
>         real matrix Diff, Sigma
>
>         Sigma = st_matrix(st_global("MH_m1"))
>         st_view(Diff=.,.,(sdiff1,sdiff2),st_global("MH_touse"))
>
>         /* compute log likelihood */
>         vcross = cross(Diff',invsym(Sigma)):*Diff
>         lnfj = -0.5*(d*ln(2*pi())+ln(det(Sigma))) :- 0.5*rowsum(vcross)
>
>         return(lnfj)
> }
: end
-------------------------------------------------------------------------------
```

The `mvnregll` program has three arguments: a scalar to store the log-likelihood values and two temporary variables containing linear predictors corresponding to each of the two dependent variables. It creates deviations `‘diff1’` and `‘diff2’` and passes them, along with other parameters, to the Mata function `mvnll_mata()` to compute the bivariate normal log-likelihood value.

The extra parameter in this model is a covariance matrix of a bivariate response. In *Simple linear regression model*, we specified an extra parameter, variance, which was a scalar, as an additional argument of the evaluator. This is not allowed with matrix parameters. They should be accessed via globals `$MH_m1`, `$MH_m2`, and so on for each matrix model parameter in the order they are specified in option `parameters()`. In our example, we have only one matrix, and we access it via `$MH_m1`. `$MH_m1` contains the temporary name of a matrix containing the current value of the covariance matrix parameter.

To demonstrate, we again use `auto.dta`. We rescale the variables to be used in our example to stabilize the results.

```
. use https://www.stata-press.com/data/r19/auto
(1978 automobile data)
. replace weight = weight/100
variable weight was int now float
(74 real changes made)
. replace length = length/10
variable length was int now float
(74 real changes made)
```

We fit a bivariate normal regression of mpg and weight on length. We specify the extra covariance parameter as a matrix model parameter, {Sigma,m}, in suboption parameters() of llevaluator(). We specify flat priors for the coefficients and an inverse-Wishart prior for the covariance matrix.

```
. set seed 14

. bayesmh mpg weight = length, llevaluator(mvnregll, parameters({Sigma,m}))
> prior({mpg:} {weight:}, flat)
> prior({Sigma,m}, iwishart(2,12,I(2)))
> mcmcsize(1000)

Burn-in ...
Simulation ...

Model summary
------------------------------------------------------------------------------
Likelihood:
  mpg weight ~ mvnregll(xb_mpg,xb_weight,{Sigma,m})

Priors:
     {mpg:length _cons} ~ 1 (flat)                                         (1)
  {weight:length _cons} ~ 1 (flat)                                         (2)
              {Sigma,m} ~ iwishart(2,12,I(2))
------------------------------------------------------------------------------
(1) Parameters are elements of the linear form xb_mpg.
(2) Parameters are elements of the linear form xb_weight.

Bayesian regression                              MCMC iterations  =      3,500
Random-walk Metropolis-Hastings sampling         Burn-in          =      2,500
                                                 MCMC sample size =      1,000
                                                 Number of obs    =         74
                                                 Acceptance rate  =      .1728
                                                 Efficiency:  min =     .02882
                                                              avg =     .05012
Log marginal-likelihood = -415.01504                          max =      .1275

------------------------------------------------------------------------------
             |                                                Equal-tailed
             |      Mean   Std. dev.     MCSE     Median  [95% cred. interval]
-------------+----------------------------------------------------------------
mpg          |
      length | -2.040162   .2009062   .037423  -2.045437  -2.369287  -1.676332
       _cons |   59.6706   3.816341   .705609   59.63619   52.54652   65.84583
-------------+----------------------------------------------------------------
weight       |
      length |   3.31773   .1461644   .026319   3.316183   3.008416   3.598753
       _cons | -32.19877    2.79005   .484962   -32.4154  -37.72904  -26.09976
-------------+----------------------------------------------------------------
   Sigma_1_1 |  11.49666   1.682975   .149035    11.3523   8.691888   14.92026
   Sigma_2_1 |  -2.33596   1.046729   .153957  -2.238129  -4.414118  -.6414916
   Sigma_2_2 |  5.830413   .9051206   .121931   5.630011   4.383648   8.000739
------------------------------------------------------------------------------
```

To reduce computation time, we used a smaller MCMC sample size of 1,000 in our example. In your analysis, you should always verify whether a smaller MCMC sample size results in precise-enough estimates before using it for final results.

We can check our results against bayesmh using the built-in multivariate normal regression after adjusting the initial values.

```
. set seed 14
. bayesmh mpg weight = length, likelihood(mvnormal({Sigma,m}))
> prior({mpg:} {weight:}, flat)
> prior({Sigma,m}, iwishart(2,12,I(2)))
> mcmcsize(1000)
> initial({mpg:} {weight:} 0)
Burn-in ...
Simulation ...
Model summary
------------------------------------------------------------------------------
Likelihood:
  mpg weight ~ mvnormal(2,xb_mpg,xb_weight,{Sigma,m})
Priors:
     {mpg:length _cons} ~ 1 (flat)                                         (1)
  {weight:length _cons} ~ 1 (flat)                                         (2)
              {Sigma,m} ~ iwishart(2,12,I(2))
------------------------------------------------------------------------------
(1) Parameters are elements of the linear form xb_mpg.
(2) Parameters are elements of the linear form xb_weight.
Bayesian multivariate normal regression          MCMC iterations  =      3,500
Random-walk Metropolis-Hastings sampling         Burn-in          =      2,500
                                                 MCMC sample size =      1,000
                                                 Number of obs    =         74
                                                 Acceptance rate  =      .1728
                                                 Efficiency:  min =     .02882
                                                              avg =     .05012
Log marginal-likelihood = -415.01504                          max =      .1275
```

	Mean	Std. dev.	MCSE	Median	Equal-tailed [95% cred.	interval]
mpg						
length	-2.040162	.2009062	.037423	-2.045437	-2.369287	-1.676332
_cons	59.6706	3.816341	.705609	59.63619	52.54652	65.84583
weight						
length	3.31773	.1461644	.026319	3.316183	3.008416	3.598753
_cons	-32.19877	2.79005	.484962	-32.4154	-37.72904	-26.09976
Sigma_1_1	11.49666	1.682975	.149035	11.3523	8.691888	14.92026
Sigma_2_1	-2.33596	1.046729	.153957	-2.238129	-4.414118	-.6414916
Sigma_2_2	5.830413	.9051206	.121931	5.630011	4.383648	8.000739

We obtain the same results.

Similarly, we can define the log-posterior evaluator. We already have the log-likelihood evaluator, which we can reuse in our log-posterior evaluator. The only additional portion is to compute the log of the inverse-Wishart prior density for the covariance parameter.

```
. program mvniWishart
  1.         version 19.5      // (or version 19 if you do not have StataNow)
  2.         args lnfj lnprior xb1 xb2
  3.         tempvar diff1 diff2 touseid
  4.         quietly generate double `diff1' = $MH_y1 - `xb1' if $MH_touse
  5.         quietly generate double `diff2' = $MH_y2 - `xb2' if $MH_touse
  6.         local d $MH_yn
  7.         local n $MH_n
  8.         quietly generate `touseid' = $MH_touse * _n
  9.         mata: st_store(st_data(.,"`touseid'"), "`lnfj'", ///
>                  mvnll_mata(`d',`n',"`diff1'","`diff2'"))
 10.         mata: st_numscalar("`lnprior'", priorWish_mata())
 11. end

. mata:
──────────────────────────────────────────────── mata (type end to exit) ─────
: real scalar priorWish_mata()
> {
>       real matrix Sigma
>       /* compute log of inverse-Wishart prior for Sigma */
>       Sigma = st_matrix(st_global("MH_m1"))
>       return(lniwishartden(12,I(2),Sigma))
> }

: end
──────────────────────────────────────────────────────────────────────────────
```

The results of the log-posterior evaluator match our earlier results.

```
. set seed 14

. bayesmh mpg weight = length, evaluator(mvniWishart, parameters({Sigma,m}))
> mcmcsize(1000)

Burn-in ...
Simulation ...

Model summary
------------------------------------------------------------------------------
Posterior:
  mpg weight ~ mvniWishart(xb_mpg,xb_weight,{Sigma,m})
------------------------------------------------------------------------------

Bayesian regression                              MCMC iterations  =      3,500
Random-walk Metropolis-Hastings sampling         Burn-in          =      2,500
                                                 MCMC sample size =      1,000
                                                 Number of obs    =         74
                                                 Acceptance rate  =      .1728
                                                 Efficiency:  min =     .02882
                                                              avg =     .05012
Log marginal-likelihood = -415.01504                          max =      .1275

------------------------------------------------------------------------------
             |                                                Equal-tailed
             |      Mean   Std. dev.     MCSE     Median  [95% cred. interval]
-------------+----------------------------------------------------------------
mpg          |
      length | -2.040162   .2009062   .037423  -2.045437  -2.369287  -1.676332
       _cons |   59.6706   3.816341   .705609   59.63619   52.54652   65.84583
-------------+----------------------------------------------------------------
weight       |
      length |   3.31773   .1461644   .026319   3.316183   3.008416   3.598753
       _cons | -32.19877    2.79005   .484962   -32.4154  -37.72904  -26.09976
-------------+----------------------------------------------------------------
   Sigma_1_1 |  11.49666   1.682975   .149035    11.3523   8.691888   14.92026
   Sigma_2_1 |  -2.33596   1.046729   .153957  -2.238129  -4.414118  -.6414916
   Sigma_2_2 |  5.830413   .9051206   .121931   5.630011   4.383648   8.000739
------------------------------------------------------------------------------
```

Sometimes, it may be useful to be able to pass options to our evaluators. For example, we used the identity I(2) matrix as a scale matrix of the inverse-Wishart distribution. Suppose that we want to check the sensitivity of our results to other choices of the scale matrix. We can pass the name of a matrix we want to use in an option. In our example, we use the vmatrix() option to pass the name of the scale matrix. We later specify this option within suboption passthruopts() of the evaluator() option. The options passed this way are stored in the $MH_passthruopts global macro.

```
. program mvniWishartV
  1.         version 19.5      // (or version 19 if you do not have StataNow)
  2.         args lnfj lnprior xb1 xb2
  3.         tempvar diff1 diff2 touseid
  4.         quietly generate double ‘diff1’ = $MH_y1 - ‘xb1’ if $MH_touse
  5.         quietly generate double ‘diff2’ = $MH_y2 - ‘xb2’ if $MH_touse
  6.         local d $MH_yn
  7.         local n $MH_n
  8.         quietly generate ‘touseid’ = $MH_touse * _n
  9.         mata: st_store(st_data(.,"‘touseid’"), "‘lnfj’", ///
>                  mvnll_mata(‘d’,‘n’,"‘diff1’","‘diff2’"))
 10.         local 0 , $MH_passthruopts
 11.         syntax, vmatrix(string)
 12.         mata: st_numscalar("‘lnprior’", priorWishV_mata("‘vmatrix’"))
 13. end

. mata:
------------------------------------------------- mata (type end to exit) -----
: real scalar priorWishV_mata(vmat)
> {
>       real matrix Sigma
>       /* compute log of inverse-Wishart prior for Sigma */
>       Sigma = st_matrix(st_global("MH_m1"))
>       return(lniwishartden(12, st_matrix(vmat), Sigma))
> }

: end
-------------------------------------------------------------------------------
```

We now define the scale matrix `V` (as the identity matrix to match our previous results) and specify `vmatrix(V)` in suboption `passthruopts()` of `evaluator()`.

```
. set seed 14
. matrix V = I(2)
. bayesmh mpg weight = length,
> evaluator(mvniWishartV, parameters({Sigma,m}) passthruopts(vmatrix(V)))
> mcmcsize(1000)
Burn-in ...
Simulation ...
Model summary
------------------------------------------------------------------------------
Posterior:
  mpg weight ~ mvniWishartV(xb_mpg,xb_weight,{Sigma,m})
------------------------------------------------------------------------------

Bayesian regression                              MCMC iterations  =      3,500
Random-walk Metropolis-Hastings sampling         Burn-in          =      2,500
                                                 MCMC sample size =      1,000
                                                 Number of obs    =         74
                                                 Acceptance rate  =      .1728
                                                 Efficiency:  min =     .02882
                                                              avg =     .05012
Log marginal-likelihood = -415.01504                          max =      .1275
```

	Mean	Std. dev.	MCSE	Median	Equal-tailed [95% cred.	interval]
mpg						
length	-2.040162	.2009062	.037423	-2.045437	-2.369287	-1.676332
_cons	59.6706	3.816341	.705609	59.63619	52.54652	65.84583
weight						
length	3.31773	.1461644	.026319	3.316183	3.008416	3.598753
_cons	-32.19877	2.79005	.484962	-32.4154	-37.72904	-26.09976
Sigma_1_1	11.49666	1.682975	.149035	11.3523	8.691888	14.92026
Sigma_2_1	-2.33596	1.046729	.153957	-2.238129	-4.414118	-.6414916
Sigma_2_2	5.830413	.9051206	.121931	5.630011	4.383648	8.000739

The results are the same as before.

Cox proportional hazards regression

Some evaluators may require additional variables, apart from the dependent and independent variables, for computation. For example, in a Cox proportional hazards model, such a variable is a failure or censoring indicator. The `coxphll` program below computes the partial log likelihood for the Cox proportional hazards regression. The failure indicator will be passed to the evaluator as an extra variable in suboption `extravars()` of option `llevaluator()` or option `evaluator()` and can be accessed from the global macro `$MH_extravars`.

```
. program coxphll
          version 19.5      // (or version 19 if you do not have StataNow)
          args lnfj xb
          tempvar negt
          quietly generate double ‘negt’ = -$MH_y1
          local d "$MH_extravars"
          sort $MH_touse ‘negt’ ‘d’
          tempvar B A sumd last L
          local byby "by $MH_touse ‘negt’ ‘d’"
          quietly {
                 gen double ‘B’ = sum(exp(‘xb’)) if $MH_touse
                 ‘byby’: gen double ‘A’ = cond(_n==_N, sum(‘xb’), .)
>                                             if ‘d’==1 & $MH_touse
                 ‘byby’: gen ‘sumd’ = cond(_n==_N, sum(‘d’), .) if $MH_touse
                 ‘byby’: gen byte ‘last’ = (_n==_N & ‘d’ == 1) if $MH_touse
                 gen double ‘L’ = ‘A’ - ‘sumd’*ln(‘B’) if ‘last’ & $MH_touse
                 replace ‘lnfj’ = 0 if $MH_touse
                 replace ‘lnfj’ = ‘L’ if ‘last’ & $MH_touse
          }
  end
```

We demonstrate the command using the survival-time `cancer` dataset. The survival-time variable is `studytime` and the failure indicator is `died`. The regressor of interest in this model is `age`. We use a fairly noninformative normal prior with a 0 mean and a variance of 100 for the regression coefficient of `age`. (The constant in the Cox proportional hazards model is not likelihood identifiable, so we omit it from this model with a noninformative prior.)

```
. use https://www.stata-press.com/data/r19/cancer, clear
(Patient survival in drug trial)

. gsort -studytime died

. set seed 14

. bayesmh studytime age, llevaluator(coxphll, extravars(died))
> prior({studytime:}, normal(0,100))
> noconstant mcmcsize(1000)

Burn-in ...
Simulation ...

Model summary
------------------------------------------------------------------------------
Likelihood:
  studytime ~ coxphll(xb_studytime)

Prior:
  {studytime:age} ~ normal(0,100)                                          (1)
------------------------------------------------------------------------------
(1) Parameter is an element of the linear form xb_studytime.

Bayesian regression                              MCMC iterations  =      3,500
Random-walk Metropolis-Hastings sampling         Burn-in          =      2,500
                                                 MCMC sample size =      1,000
                                                 Number of obs    =         48
                                                 Acceptance rate  =      .4066
Log marginal-likelihood = -103.04797             Efficiency       =      .3568

------------------------------------------------------------------------------
             |                                                Equal-tailed
   studytime |      Mean   Std. dev.     MCSE     Median  [95% cred. interval]
-------------+----------------------------------------------------------------
         age |   .076705   .0330669   .001751    .077936   .0099328   .1454275
------------------------------------------------------------------------------
```

We specified the failure indicator `died` in suboption `extravars()` of `llevaluator()`. We again used a smaller value for the MCMC sample size only to reduce computation time.

For the log-posterior evaluator, we add the log of the normal prior of the age coefficient to the log-likelihood value to obtain the final log-posterior value. We did not need to specify the loop in the log-prior computation in this example, but we did this to be general, in case more than one regressor is included in the model.

```
. program coxphnormal
  1.         version 19.5      // (or version 19 if you do not have StataNow)
  2.         args lnfj lnprior xb
  3.         /* compute log likelihood */
  4.         quietly coxphll `lnfj' `xb'
  5.         /* compute log priors of regression coefficients */
  6.         scalar `lnprior' = 0
  7.         forvalues i = 1/$MH_bn {
  8.             scalar `lnprior' = `lnprior' + lnnormalden($MH_b[1,`i'], 10)
  9.         }
 10. end
```

As expected, we obtain the same results as previously.

```
. set seed 14

. bayesmh studytime age, evaluator(coxphnormal, extravars(died))
> noconstant mcmcsize(1000)

Burn-in ...
Simulation ...

Model summary
------------------------------------------------------------------------------
Posterior:
  studytime ~ coxphnormal(xb_studytime)
------------------------------------------------------------------------------

Bayesian regression                              MCMC iterations  =      3,500
Random-walk Metropolis-Hastings sampling         Burn-in          =      2,500
                                                 MCMC sample size =      1,000
                                                 Number of obs    =         48
                                                 Acceptance rate  =      .4066
Log marginal-likelihood = -103.04797             Efficiency       =      .3568

------------------------------------------------------------------------------
             |                                                Equal-tailed
   studytime |      Mean   Std. dev.     MCSE     Median  [95% cred. interval]
-------------+----------------------------------------------------------------
         age |   .076705   .0330669   .001751    .077936   .0099328   .1454275
------------------------------------------------------------------------------
```

Random-intercept linear regression model

In the next few examples, we demonstrate the use of evaluators for fitting random-intercept models. We first reuse the likelihood evaluator `normalreg`, defined in *Simple linear regression model*.

We consider `pig.dta` and fit a linear regression of `weight` on `week` with a random intercept at the levels of the `id` variable, which we include as `{U[id]}` in our regression specification; see *Random effects* in [BAYES] **bayesmh**. To fit this model with `bayesmh` using the evaluator version, we specify the `llevaluator()` option and pass the error variance parameter `{var}` using the `parameters()` suboption. We choose to drop the constant term from the regression equation and assign a normal prior centered at parameter `{weight:_cons}` for the random effects `{U[id]}`. This will improve sampling efficiency. To further improve sampling efficiency, in the presence of random effects, `bayesmh` automatically blocks random-effects parameters by applying the `block({U[id]}, reffects)` option. To complete the model specification, we also add priors for the error variance `{var}` and the variance of random intercept `{var_U}`. In addition, the `init()` and `rseed()` options are used for reproducibility.

```
. use https://www.stata-press.com/data/r19/pig
(Longitudinal analysis of pig weights)

. bayesmh weight week U[id], noconstant
>                            llevaluator(normalreg, parameters({var}))
>                            prior({U[id]}, normal({weight:_cons}, {var_U}))
>                            prior({weight:}, normal(0, 10000))
>                            prior({var_U}, igamma(0.01, 0.01)) block({var_U})
>                            prior({var}, igamma(0.01, 0.01)) block({var})
>                            init({weight:} 0 {var} 1) rseed(19)

Burn-in 2500 aaaaaaaaa1000aaaaaaaaa2000aaaaa done
Simulation 10000 .........1000.........2000.........3000.........4000.........5
> 000.........6000.........7000.........8000.........9000.........10000 done

Model summary
-------------------------------------------------------------------------------
Likelihood:
  weight ~ normalreg(xb_weight,{var})

Priors:
   {weight:week} ~ normal(0,10000)                                          (1)
         {U[id]} ~ normal({weight:_cons},{var_U})                           (1)
           {var} ~ igamma(0.01,0.01)
  {weight:_cons} ~ normal(0,10000)

Hyperprior:
  {var_U} ~ igamma(0.01,0.01)
-------------------------------------------------------------------------------
(1) Parameters are elements of the linear form xb_weight.

Bayesian regression                              MCMC iterations  =     12,500
Random-walk Metropolis-Hastings sampling         Burn-in          =      2,500
                                                 MCMC sample size =     10,000
                                                 Number of obs    =        432
                                                 Acceptance rate  =      .3181
                                                 Efficiency:  min =     .01933
                                                              avg =     .08653
Log marginal-likelihood                                       max =      .1747

-------------------------------------------------------------------------------
             |                                                Equal-tailed
             |      Mean   Std. dev.     MCSE     Median  [95% cred. interval]
-------------+-----------------------------------------------------------------
weight       |
        week |  6.210361   .0357788   .002573   6.210529    6.13508   6.283522
       _cons |  19.35363   .6087435   .031972   19.34191   18.26791   20.58926
-------------+-----------------------------------------------------------------
         var |  4.426645   .3247232   .007768   4.409389   3.835124   5.102071
       var_U |  15.92844   3.801803   .111729    15.3975   10.11643   24.55804
-------------------------------------------------------------------------------
```

Because `bayesmh` automatically includes the random-effects parameters in the linear predictor (the program argument `xb`), we did not need to modify the `normalreg` evaluator to accommodate random effects. Some evaluators may need direct access to random effects. Below, for demonstration, we provide an equivalent evaluator, `normalre`, that is an extended version of `normalreg`, where we manually build the linear form of the regression model by using the model parameters, including the random effects.

We assume that the model has one predictor. The name of the predictor variable is available in the `MH_x1` global macro. There is only one regression coefficient, `{mpg:weight}`, and its value is available in the `$MH_b` matrix. To access the random intercepts within our program, we include `{U}` in the `reparameters()` suboption of the `llevaluator()` option of `bayesmh`. The list of arguments in `normalre` thus includes a local macro `U` with the name of a temporary variable, `‘U’`, containing the current values of the random intercept across the observations.

The linear form, which we compute and store in a temporary variable `‘xb2’`, matches the provided temporary variable `‘xb’`.

```
. program normalre
  1.     version 19.5      // (or version 19 if you do not have StataNow)
  2.     args lnfj xb var U
  3.     tempvar xb2
  4.     tempname mb sd
  5.     scalar ‘sd’ = sqrt(‘var’)
  6.     /* retrieve regression coefficients for covariates */
.        matrix ‘mb’ = $MH_b
  7.     /* compute linear form */
.        quietly generate double ‘xb2’ = $MH_x1*‘=‘mb’[1,1]’ + ‘U’ if $MH_touse
  8.     /* compute log-likelihood */
.        quietly replace ‘lnfj’ = lnnormalden($MH_y,‘xb2’,‘sd’) if $MH_touse
  9. end
```

The only change in the `bayesmh` specification for the new evaluator is the inclusion of the `reparameters({U})` suboption.

```
. bayesmh weight week U[id], noconstant
>                            llevaluator(normalre, parameters({var}) reparameters({U}))
>                            prior({U[id]}, normal({weight:_cons}, {var_U}))
>                            prior({weight:}, normal(0, 10000))
>                            prior({var_U}, igamma(0.01, 0.01)) block({var_U})
>                            prior({var}, igamma(0.01, 0.01)) block({var})
>                            init({weight:} 0 {var} 1) rseed(19)

Burn-in 2500 aaaaaaaaa1000aaaaaaaaa2000aaaaa done
Simulation 10000 .........1000.........2000.........3000.........4000.........5
> 000.........6000.........7000.........8000.........9000.........10000 done

Model summary
------------------------------------------------------------------------------
Likelihood:
  weight ~ normalre(xb_weight,{var},{U[id]})

Priors:
   {weight:week} ~ normal(0,10000)                                          (1)
         {U[id]} ~ normal({weight:_cons},{var_U})                           (1)
           {var} ~ igamma(0.01,0.01)
  {weight:_cons} ~ normal(0,10000)

Hyperprior:
  {var_U} ~ igamma(0.01,0.01)
------------------------------------------------------------------------------
(1) Parameters are elements of the linear form xb_weight.
```

```
Bayesian regression                              MCMC iterations  =     12,500
Random-walk Metropolis-Hastings sampling         Burn-in          =      2,500
                                                 MCMC sample size =     10,000
                                                 Number of obs    =        432
                                                 Acceptance rate  =      .3181
                                                 Efficiency:  min =     .01933
                                                              avg =     .08653
Log marginal-likelihood                                       max =      .1747
```

	Mean	Std. dev.	MCSE	Median	Equal-tailed [95% cred.	interval]
weight						
week	6.210361	.0357788	.002573	6.210529	6.13508	6.283522
_cons	19.35363	.6087435	.031972	19.34191	18.26791	20.58926
var	4.426645	.3247232	.007768	4.409389	3.835124	5.102071
var_U	15.92844	3.801803	.111729	15.3975	10.11643	24.55804

As expected, the estimation results produced by bayesmh using the two equivalent evaluators are the same.

Evaluators with predictions

If you want to fit a model with bayesmh using an evaluator and then compute predictions by using bayespredict (see [BAYES] **bayespredict**), your evaluator should also be able to generate samples from the outcome distribution of your likelihood model.

To implement on-demand predictions, your evaluator needs to check the global macro $MH_predict, which will be set by bayespredict during its run time. If it is set to 1, then the evaluator needs to generate an outcome sample and store the generated values in the temporary variables $MH_predict_y1 for the first outcome, in $MH_predict_y2 for the second outcome, and so on.

Below, we add support for predictions to the normalreg log-likelihood evaluator from *Simple linear regression model*. The log-posterior evaluators can be extended similarly. In this case, we have only one outcome variable, $MH_y1. If $MH_predict is set to 1, we need to generate a random sample from the data distribution conditional on the current set of model parameters, which is a normal distribution with the provided mean ‘xb’ and variance ‘var’, and store the sample in $MH_predict_y1. In addition, we need to store the expected value for the outcome in $MH_predict_mu1. For a normal distribution, the expected outcome is given by the linear form ‘xb’.

When `$MH_predict` is set, we do not need to compute and return the log-likelihood values because the evaluator is used purely for prediction in that case.

```
. program normalpr
  1.      version 19.5      // (or version 19 if you do not have StataNow)
  2.      args lnfj xb var
  3.      tempname sd
  4.      scalar ‘sd’ = sqrt(‘var’)
  5.      if $MH_predict {
  6.          quietly replace $MH_predict_mu1 = ‘xb’ if $MH_touse
  7.          quietly replace $MH_predict_y1 = rnormal(‘xb’,‘sd’) if $MH_touse
  8.      }
  9.      else {
 10.          quietly replace ‘lnfj’ = lnnormalden($MH_y,‘xb’,‘sd’) if $M
> H_touse
 11.      }
 12. end
```

Working with the `pig.dta` dataset of the previous section, we use our `normalpr` evaluator to fit a simple linear regression of `weight` on `week`. To indicate that the evaluator implements predictions, we specify the `predict` suboption within the `llevaluator()` option. We also add the `saving()` option to the specification to save the simulation results in a permanent dataset, because this is required to compute predictions later.

```
. bayesmh weight week, llevaluator(normalpr, parameters({var}) predict)
>                      prior({weight:}, normal(0, 10000))
>                      prior({var}, igamma(0.01, 0.01)) block({var})
>                      init({weight:} 0 {var} 1) rseed(19) saving(bayesmhsim)

Burn-in ...
Simulation ...

Model summary
------------------------------------------------------------------------------
Likelihood:
  weight ~ normalpr(xb_weight,{var})

Priors:
  {weight:week _cons} ~ normal(0,10000)                                    (1)
                {var} ~ igamma(0.01,0.01)
------------------------------------------------------------------------------
(1) Parameters are elements of the linear form xb_weight.

Bayesian regression                                MCMC iterations  =     12,500
Random-walk Metropolis-Hastings sampling           Burn-in          =      2,500
                                                   MCMC sample size =     10,000
                                                   Number of obs    =        432
                                                   Acceptance rate  =      .3321
                                                   Efficiency:  min =      .1201
                                                                avg =      .1622
Log marginal-likelihood = -1270.8744                            max =      .2348

------------------------------------------------------------------------------
             |                                                Equal-tailed
             |      Mean   Std. dev.     MCSE     Median  [95% cred. interval]
-------------+----------------------------------------------------------------
weight       |
        week |  6.213014   .0841125   .002317   6.214818   6.040679   6.380262
       _cons |  19.34526   .4672249   .013482   19.33243   18.44014   20.29782
-------------+----------------------------------------------------------------
         var |  19.41488   1.388441   .028654   19.38093   16.80829   22.42214
------------------------------------------------------------------------------

file bayesmhsim.dta saved.
```

We can now use the `bayespredict` command to compute predicted posterior means for the outcome `weight`. The predicted values are stored in a new variable, `prweight`. For a quick comparison of the observed and predicted outcomes, we use their means and standard deviations. (We also drop the newly generated variable and erase the simulation file to clean up.)

```
. bayespredict prweight if e(sample), mean
Computing predictions ...
. summarize prweight weight if e(sample)
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
    prweight |        432    50.41117    16.05947   25.44927   75.33538
      weight |        432    50.40509    16.64113         20         88
. drop prweight
. rm bayesmhsim.dta
```

The estimated mean of `prweight`, 50.41, is close to that of `weight`, but its standard deviation, 16.06, is slightly less than that of `weight`, 16.64.

We can also use the `normalpr` evaluator with random-effects models. For example, we can refit the random-intercept model from *Random-intercept linear regression model*.

```
. bayesmh weight week U[id], noconstant
>                            llevaluator(normalpr, parameters({var}) predict)
>                            prior({U[id]}, normal({weight:_cons}, {var_U}))
>                            prior({weight:}, normal(0, 10000))
>                            prior({var_U}, igamma(0.01, 0.01)) block({var_U})
>                            prior({var}, igamma(0.01, 0.01)) block({var})
>                            init({weight:} 0 {var} 1) rseed(19) saving(bayesmhsim)
Burn-in 2500 aaaaaaaaa1000aaaaaaaaa2000aaaaa done
Simulation 10000 .........1000.........2000.........3000.........4000.........5
> 000.........6000.........7000.........8000.........9000.........10000 done
Model summary
------------------------------------------------------------------------------
Likelihood:
  weight ~ normalpr(xb_weight,{var})
Priors:
   {weight:week} ~ normal(0,10000)                                         (1)
         {U[id]} ~ normal({weight:_cons},{var_U})                          (1)
           {var} ~ igamma(0.01,0.01)
  {weight:_cons} ~ normal(0,10000)
Hyperprior:
  {var_U} ~ igamma(0.01,0.01)
------------------------------------------------------------------------------
(1) Parameters are elements of the linear form xb_weight.
```

```
Bayesian regression                              MCMC iterations  =     12,500
Random-walk Metropolis-Hastings sampling         Burn-in          =      2,500
                                                 MCMC sample size =     10,000
                                                 Number of obs    =        432
                                                 Acceptance rate  =      .3181
                                                 Efficiency:  min =     .01933
                                                              avg =     .08653
Log marginal-likelihood                                       max =      .1747

------------------------------------------------------------------------------
             |                                                Equal-tailed
             |      Mean   Std. dev.     MCSE     Median  [95% cred. interval]
-------------+----------------------------------------------------------------
weight       |
        week |  6.210361   .0357788   .002573   6.210529    6.13508   6.283522
       _cons |  19.35363   .6087435   .031972   19.34191   18.26791   20.58926
-------------+----------------------------------------------------------------
         var |  4.426645   .3247232   .007768   4.409389   3.835124   5.102071
       var_U |  15.92844   3.801803   .111729    15.3975   10.11643   24.55804
------------------------------------------------------------------------------
file bayesmhsim.dta saved.
```

The estimation results match the results from the earlier random-intercept model.

```
. bayespredict prweight if e(sample), mean
Computing predictions ...
. summarize prweight weight if e(sample)
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
    prweight |        432    50.40606    16.49598    18.3743   84.36924
      weight |        432    50.40509    16.64113         20         88
. drop prweight
. rm bayesmhsim.dta
```

Again, the observed and predicted outcome generally agree, but the predicted one has slightly less variability, as indicated by the estimated standard deviations.

Global macros

Global macros	Description
$MH_n	number of observations
$MH_yn	number of dependent variables
$MH_touse	variable containing 1 for the observations to be used; 0 otherwise
$MH_w	variable containing weight associated with the observations
$MH_extravars	*varlist* specified in extravars()
$MH_passthruopts	options specified in passthruopts()
One outcome	
$MH_y1	name of the dependent variable
$MH_x1	name of the first independent variable
$MH_x2	name of the second independent variable
...	
$MH_xn	number of independent variables
$MH_xb	name of a temporary variable containing the linear combination
Multiple outcomes	
$MH_y1	name of the first dependent variable
$MH_y2	name of the second dependent variable
...	
$MH_y1x1	name of the first independent variable modeling y1
$MH_y1x2	name of the second independent variable modeling y1
...	
$MH_y1xn	number of independent variables modeling y1
$MH_y1xb	name of a temporary variable containing the linear combination modeling y1
$MH_y2x1	name of the first independent variable modeling y2
$MH_y2x2	name of the second independent variable modeling y2
...	
$MH_y2xn	number of independent variables modeling y2
$MH_y2xb	name of a temporary variable containing the linear combination modeling y2
...	
Scalar and matrix parameters	
$MH_b	name of a temporary vector of coefficients; stripes are properly named after the name of the coefficients
$MH_bn	number of coefficients
$MH_p	name of a temporary vector of additional scalar model parameters, if any; stripes are properly named
$MH_pn	number of additional scalar model parameters
$MH_m1	name of a temporary matrix of the first matrix parameter, if any
$MH_m2	name of a temporary matrix of the second matrix parameter, if any
...	
$MH_mn	number of matrix model parameters

Global macros, cont.	Description, cont.
Random effects	
$MH_RE_tempvars	names of temporary variables containing values for random-effects parameters specified in option reparameters()
$MH_RE#	name of temporary variable for the #th random-effects parameter
$MH_RE#_name	name of random-effects parameter corresponding to $MH_RE#
$MH_RE#_levelspec	level specification of random-effects parameter $MH_RE#_name
Prediction	
$MH_predict	prediction flag set to 1 by bayespredict during its run time if suboption predict was specified with an evaluator; 0 otherwise
$MH_predict_y1	name of temporary variable containing predictions for y1
$MH_predict_y2	name of temporary variable containing predictions for y2
...	
$MH_predict_mu1	name of temporary variable containing expected value of y1
$MH_predict_mu2	name of temporary variable containing expected value of y2
...	

Stored results

In addition to the results stored by `bayesmh`, `bayesmh, evaluator()` and `bayesmh, llevaluator()` store the following in `e()`:

Macros

`e(evaluator)`	name of evaluator program (one equation)
`e(evaluator#)`	name of evaluator program for the #th equation
`e(evalparams)`	evaluator parameters (one equation)
`e(evalparams#)`	evaluator parameters for the #th equation
`e(evalreparams)`	evaluator random-effects parameters (one equation)
`e(evalreparams#)`	evaluator random-effects parameters for the #th equation
`e(extravars)`	extra variables (one equation)
`e(extravars#)`	extra variables for the #th equation
`e(passthruopts)`	pass-through options (one equation)
`e(passthruopts#)`	pass-through options for the #th equation

Reference

Marchenko, Y. V. 2015. Bayesian modeling: Beyond Stata's built-in models. *The Stata Blog: Not Elsewhere Classified.* https://blog.stata.com/2015/05/26/bayesian-modeling-beyond-statas-built-in-models/.

Also see

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

[BAYES] **Bayesian postestimation** — Postestimation tools after Bayesian estimation

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

[BAYES] **Glossary**

For suggested citations, see the FAQ on citing Stata documentation.