Title

stata.com

bayesmh evaluators — User-defined evaluators with bayesmh

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*)]

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

paramdef [*paramdef* [...]]

The syntax of *paramdef* is

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

where 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**.

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_*`)`	prior for model parameters; required with log-likelihood evaluator and may be repeated
`likelihood(`*modelspec*`)`	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
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 *Program 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 *Program 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**.

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

Remarks and examples

stata.com

Remarks are presented under the following headings:

Program 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. Your program is expected to calculate and return an overall log-posterior or a log-likelihood density value.

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

```
program progname
    args lnden xb1 [xb2 ...] [modelparams]
    ... computations ...
    scalar 'lnden' = ...
end
```

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

xb# contains the name of a temporary variable containing the linear predictor from the #th equation; and

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 containing current values. 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().

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

After you write a program evaluator, 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(), and passthruopts().

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 program evaluator. 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 program evaluator. For example, these options may contain fixed values of model parameters and hyperparameters. See *Multivariate normal regression model* for an example.

`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 by the program evaluator. 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 18.0
  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
```

The program accepts three parameters: a temporary name `` `lnp' `` of a scalar to contain the log-posterior value, a temporary name `` `xb' `` of the variable that contains the linear predictor, and a temporary name `` `var' `` of a scalar that contains the values of the variance parameter.

The first part of the program calculates the overall 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. The third and final part of the program computes the values of the posterior density as the sum of the overall log likelihood and the log of the prior.

The substantial portion of this program is the computation of the overall log likelihood. The global macro `$MH_y` contains the name of the dependent variable, `$MH_touse` contains a temporary marker

variable identifying observations to be used in computations, and $MH_n contains the total number of observations in the sample identified by the $MH_touse variable.

We used the built-in function lnnormalden() to compute observation-specific log likelihood and used summarize to obtain the overall value. Whenever a temporary variable is needed for calculations, such as 'lnfj' in our program, it is important 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, if, and in affect this variable. 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.

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/r18/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.

```
. regress mpg weight
```

Source	SS	df	MS	Number of obs	=	74
				F(1, 72)	=	134.62
Model	1591.99021	1	1591.99021	Prob > F	=	0.0000
Residual	851.469254	72	11.8259619	R-squared	=	0.6515
				Adj R-squared	=	0.6467
Total	2443.45946	73	33.4720474	Root MSE	=	3.4389

mpg	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
weight	-.6008687	.0517878	-11.60	0.000	-.7041058	-.4976315
_cons	39.44028	1.614003	24.44	0.000	36.22283	42.65774

```
. display e(rmse)^2
11.825962
```

We 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
```

	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

We can compare our results with `bayesmh` that 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 extract the portion of the `normaljeffreys` program computing the overall log likelihood into a separate program and call it `normalreg`.

```
. program normalreg
  1.         version 18.0
  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 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.

Logistic regression model

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

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

```
. program logitll
  1.         version 18.0
  2.         args lnf xb
  3.         tempvar lnfj
  4.         quietly generate `lnfj' = ln(invlogit( `xb'))
>                                       if $MH_y == 1 & $MH_touse
  5.         quietly replace `lnfj'  = ln(invlogit(-`xb'))
>                                       if $MH_y == 0 & $MH_touse
  6.         quietly summarize `lnfj', meanonly
  7.         if r(N) < $MH_n {
  8.             scalar `lnf' = .
  9.             exit
 10.         }
 11.         scalar `lnf' = r(sum)
 12. 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/r18/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 = -41.626028                         max =      .1068
```

foreign	Mean	Std. dev.	MCSE	Median	Equal-tailed [95% cred.	interval]
mpg	.16716	.0545771	.00167	.1644019	.0669937	.2790017
_cons	-4.560637	1.261675	.041387	-4.503921	-7.107851	-2.207665

The results from the program-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.

```
. set seed 14
. bayesmh foreign mpg, evaluator(logitll)
Burn-in ...
Simulation ...
Model summary
------------------------------------------------------------------------------
Posterior:
  foreign ~ logitll(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 = -41.626028                         max =      .1068
------------------------------------------------------------------------------
                                                                Equal-tailed
     foreign |      Mean   Std. dev.     MCSE     Median  [95% cred. interval]
-------------+----------------------------------------------------------------
         mpg |     .16716   .0545771    .00167   .1644019    .0669937   .2790017
       _cons |  -4.560637   1.261675   .041387  -4.503921  -7.107851  -2.207665
------------------------------------------------------------------------------
```

Multivariate normal regression model

Here we demonstrate how to write a program evaluator 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 18.0
  2.         args lnf xb1 xb2
  3.         tempvar diff1 diff2
  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.         mata: st_numscalar("`lnf'", mvnll_mata(`d',`n',"`diff1'","`diff2'"))
  9. end
.
. mata:
------------------------------------------------- mata (type end to exit) -----
: real scalar mvnll_mata(real scalar d, n, string scalar sdiff1, sdiff2)
> {
>         real matrix Diff
>         real scalar trace, lnf
>         real matrix Sigma
>
>         Sigma = st_matrix(st_global("MH_m1"))
>         st_view(Diff=.,.,(sdiff1,sdiff2),st_global("MH_touse"))
>
>         /* compute log likelihood */
>         trace = trace(cross(cross(Diff',invsym(Sigma))',Diff'))
>         lnf = -0.5*n*(d*ln(2*pi())+ln(det(Sigma)))-0.5*trace
>
>         return(lnf)
> }
: 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 parameters 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/r18/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
```

	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

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 18.0
  2.         args lnp xb1 xb2
  3.         tempvar diff1 diff2
  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.         mata:
>             st_numscalar("`lnp'", mvniWish_mata(`d',`n',"`diff1'","`diff2'"))
  9. end

.
. mata:
------------------------------------------------- mata (type end to exit) -----
: real scalar mvniWish_mata(real scalar d, n, string scalar sdiff1, sdiff2)
> {
>         real scalar lnf, lnprior
>         real matrix Sigma
>
>         /* compute log likelihood */
>         lnf = mvnll_mata(d,n,sdiff1,sdiff2)
>         /* compute log of inverse-Wishart prior for Sigma */
>         Sigma = st_matrix(st_global("MH_m1"))
>         lnprior = lniwishartden(12,I(2),Sigma)
>         return(lnf + lnprior)
> }
: 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 18.0
  2.         args lnp xb1 xb2
  3.         tempvar diff1 diff2
  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.         local 0 , $MH_passthruopts
  9.         syntax, vmatrix(string)
 10.         mata: st_numscalar("`lnp'",
>               mvniWishV_mata(`d',`n',"`diff1'","`diff2'","`vmatrix'"))
 11. end
```

```
. mata:
------------------------------------------------- mata (type end to exit) -----
: real scalar mvniWishV_mata(real scalar d, n, string scalar sdiff1, sdiff2,
> vmat)
> {
>         real scalar lnf, lnprior
>         real matrix Sigma
>
>         /* compute log likelihood */
>         lnf = mvnll_mata(d,n,sdiff1,sdiff2)
>         /* compute log of inverse-Wishart prior for Sigma */
>         Sigma = st_matrix(st_global("MH_m1"))
>         lnprior = lniwishartden(12,st_matrix(vmat),Sigma)
>         return(lnf + lnprior)
> }
: 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

------------------------------------------------------------------------------
             |                                                 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
------------------------------------------------------------------------------
```

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 variable is a failure or censoring indicator. The `coxphll` program below computes 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
  1.         version 18.0
  2.         args lnf xb
  3.         tempvar negt
  4.         quietly generate double `negt' = -$MH_y1
  5.         local d "$MH_extravars"
  6.         sort $MH_touse `negt' `d'
  7.         tempvar B A sumd last L
  8.         local byby "by $MH_touse `negt' `d'"
  9.         quietly {
 10.                gen double `B' = sum(exp(`xb')) if $MH_touse
 11.                `byby': gen double `A' = cond(_n==_N, sum(`xb'), .)
>                                         if `d'==1 & $MH_touse
 12.                `byby': gen `sumd' = cond(_n==_N, sum(`d'), .) if $MH_touse
 13.                `byby': gen byte `last' = (_n==_N & `d' == 1) if $MH_touse
 14.                gen double `L' = `A' - `sumd'*ln(`B') if `last' & $MH_touse
 15.                quietly count if $MH_touse & `last'
 16.                local n = r(N)
 17.                summarize `L' if `last' & $MH_touse, meanonly
 18.         }
 19.         if r(N) < `n' {
 20.             scalar `lnf' = .
 21.             exit
 22.         }
 23.         scalar `lnf' = r(sum)
 24. 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 zero 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/r18/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 18.0
  2.         args lnp xb
.         /* compute log likelihood */
.         tempname lnf
  3.         scalar `lnf' = .
  4.         quietly coxphll `lnf' `xb'
.         /* compute log priors of regression coefficients */
.         tempname lnprior
  5.         scalar `lnprior' = 0
  6.         forvalues i = 1/$MH_bn {
  7.             scalar `lnprior' = `lnprior' + lnnormalden($MH_b[1,`i'], 10)
  8.         }
  9.         /* compute log posterior */
.         scalar `lnp' = `lnf' + `lnprior'
 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
------------------------------------------------------------------------------
```

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

Stored results

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

Macros

`e(evaluator)`	program evaluator (one equation)
`e(evaluator#)`	program evaluator for the #th equation
`e(evalparams)`	evaluator parameters (one equation)
`e(evalparams#)`	evaluator 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.* http://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 for bayesmh and the bayes prefix

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

[BAYES] **Glossary**