

# Bayesian estimation with `bayesmh`

Yaojin Sun

Senior Statistician and Software Developer

April 28, 2026

# Housekeeping

- ⦿ **This webinar is being recorded.** A link to the recording will be sent out after the presentation.
- ⦿ **For questions on the content, please use the Zoom Q&A box** (not the chat panel). My colleagues will answer your questions there.

## A Note on Sources

- ⦿ **This presentation draws heavily on the Stata official manual**, in particular `[BAYES] bayesmh` and `[BAYES] bayesmh evaluators`, for both the methods and the worked examples.

# Outline

- 1 Quickstart
- 2 Scope
- 3 Writing Your Own Evaluator
- 4 Matched-Condition Replication Protocol
- 5 No Matched-Condition Protocol
- 6 Summary

# Stata's Bayesian Suite

## What the Bayesian module contains.

- ⦿ The `bayesmh` command for fitting arbitrary Bayesian models.
- ⦿ The `bayes:` prefix for fitting supported Bayesian models with familiar Stata estimation syntax.
- ⦿ A full set of Bayesian post-estimation commands ( `bayesstats` , `bayestest` , `bayespredict` ) and graphical diagnostics ( `bayesgraph` ).

## How `bayesmh` and `bayes:` relate.

- ⦿ `bayesmh` is the foundation and core engine of Stata's Bayesian framework.
- ⦿ Most `bayes:` prefix commands are built directly on `bayesmh` , supplying a convenience syntax that assembles the underlying `bayesmh` call for you.
- ⦿ Both interfaces serve users at every experience level. `bayes:` is the faster entry point when your model is on the supported list; `bayesmh` is where you go when you need the extra flexibility.

## The MH Loop Inside `bayesmh`

`bayesmh` draws from the posterior with adaptive random-walk Metropolis–Hastings. Each iteration runs the same four steps:

- 1 **Propose.** Draw a candidate parameter vector  $\theta^*$  from a proposal distribution that adapts during burn-in.
- 2 **Score.** Compute the log-posterior at the candidate:  $\log p(y \mid \theta^*) + \log p(\theta^*)$ .
- 3 **Decide.** Accept  $\theta^*$  with probability  $\alpha = \min\left(1, \frac{p(\theta^*|y)}{p(\theta_{t-1}|y)}\right)$ ; otherwise keep  $\theta_{t-1}$ .
- 4 **Record.** After burn-in, store each retained draw (respecting thinning) as the posterior sample.

# Built-in Likelihoods

Stata already ships a large catalog of likelihood families for `bayesmh`. Reach for the built-in first; only drop to `llf()` or `llevauator()` when your model is not on the list.

<i>model</i>	Description
<b>Model</b>	
<code>normal(var)</code>	normal regression with variance <i>var</i>
<code>t(sigma2, df)</code>	<i>t</i> regression with squared scale <i>sigma2</i> and degrees of freedom <i>df</i>
<code>lognormal(var)</code>	lognormal regression with variance <i>var</i>
<code>lnormal(var)</code>	synonym for <code>lognormal()</code>
<code>exponential</code>	exponential regression
<code>asymlaplaceq(sigma, tau)</code>	asymmetric Laplace (quantile) regression with scale <i>sigma</i> and quantile <i>tau</i>
<code>mvnormal(Sigma)</code>	multivariate normal regression with covariance matrix <i>Sigma</i>
<code>probit</code>	probit regression
<code>logit</code>	logistic regression
<code>logistic</code>	logistic regression; synonym for <code>logit</code>
<code>binomial(n)</code>	binomial regression with logit link and number of trials <i>n</i>
<code>binlogit(n)</code>	synonym for <code>binomial()</code>
<code>oprobit</code>	ordered probit regression
<code>ologit</code>	ordered logistic regression
<code>poisson</code>	Poisson regression
<b>Survival</b>	
<code>stexponential</code>	exponential survival regression
<code>stgamma(lns)</code>	gamma survival regression with log-scale parameter <i>lns</i>
<code>stloglogistic(lns)</code>	loglogistic survival regression with log-scale parameter <i>lns</i>
<code>stlognormal(lnsd)</code>	lognormal survival regression with log-standard-deviation parameter <i>lnsd</i>
<code>stweibull(lnp)</code>	Weibull survival regression with log-shape parameter <i>lnp</i>
<code>llf(subexpr)</code>	substitutable expression for observation-level log-likelihood function

Where to find it: the `[BAYES] bayesmh` entry in the Stata PDF manual, under the `likelihood()` option.

# Built-in Priors

Every declared parameter needs a prior. Stata ships a large library of univariate and multivariate prior distributions that plug directly into `prior()`.

<i>priorlist</i>	Description
<b>Model</b>	
<code>normal(mu, var)</code>	normal with mean <i>mu</i> and variance <i>var</i>
<code>t(mu, sigma2, df)</code>	location-scale <i>t</i> with mean <i>mu</i> , squared scale <i>sigma2</i> , and degrees of freedom <i>df</i>
<code>lognormal(mu, var)</code>	lognormal with mean <i>mu</i> and variance <i>var</i>
<code>lnormal(mu, var)</code>	synonym for <code>lognormal()</code>
<code>uniform(a, b)</code>	uniform on $(a, b)$
<code>gamma(alpha, beta)</code>	gamma with shape <i>alpha</i> and scale <i>beta</i>
<code>igamma(alpha, beta)</code>	inverse gamma with shape <i>alpha</i> and scale <i>beta</i>
<code>exponential(beta)</code>	exponential with scale <i>beta</i>
<code>beta(a, b)</code>	beta with shape parameters <i>a</i> and <i>b</i>
<code>laplace(mu, beta)</code>	Laplace with mean <i>mu</i> and scale <i>beta</i>
<code>cauchy(loc, beta)</code>	Cauchy with location <i>loc</i> and scale <i>beta</i>
<code>halfcauchy(loc, beta)</code>	half-Cauchy with location <i>loc</i> and scale <i>beta</i>
<code>chi2(df)</code>	central $\chi^2$ with degrees of freedom <i>df</i>
<code>rayleigh(beta)</code>	Rayleigh distribution with scale <i>beta</i>
<code>pareto(alpha, beta)</code>	Pareto with shape <i>alpha</i> and scale <i>beta</i>
<code>jeffreys</code>	Jeffreys prior for variance of a normal distribution
<code>mvnormal(d, mean, Sigma)</code>	multivariate normal of dimension <i>d</i> with mean vector <i>mean</i> and covariance matrix <i>Sigma</i> ; <i>mean</i> can be a matrix name or a list of <i>d</i> means separated by comma: <i>mu</i> <sub>1</sub> , <i>mu</i> <sub>2</sub> , ..., <i>mu</i> <sub><i>d</i></sub>
<code>mvnormal0(d, Sigma)</code>	multivariate normal of dimension <i>d</i> with zero mean vector and covariance matrix <i>Sigma</i>
<code>mvn0(d, Sigma)</code>	synonym for <code>mvnormal0()</code>
<code>mvnexchangeable(d, mean, var, rho)</code>	multivariate normal of dimension <i>d</i> with means <i>mean</i> and exchangeable covariance matrix with diagonal <i>var</i> and off-diagonal <i>var</i> × <i>rho</i>
<code>mvn0exchangeable(d, var, rho)</code>	as <code>mvnexchangeable()</code> but with zero mean vector
<code>mvnindependent(d, mean, vars)</code>	multivariate normal of dimension <i>d</i> with means <i>mean</i> and diagonal covariance matrix; <i>vars</i> can be a Stata vector of dimension <i>d</i> with fixed variances or a list of <i>d</i> variances (parameters or fixed values) separated by comma: <i>var</i> <sub>1</sub> , <i>var</i> <sub>2</sub> , ..., <i>var</i> <sub><i>d</i></sub>
<code>mvn0independent(d, vars)</code>	as <code>mvnindependent()</code> but with zero mean vector
<code>mvnidentity(d, mean, var)</code>	multivariate normal of dimension <i>d</i> with means <i>mean</i> and identity covariance matrix with equal variances <i>var</i>
<code>mvn0identity(d, var)</code>	as <code>mvnidentity()</code> but with zero mean vector
<code>mvnscaled(d, mean, A, {var})</code>	multivariate normal of dimension <i>d</i> with mean vector <i>mean</i> and covariance matrix $\{var\}A$ ; <i>mean</i> can be a matrix name or a list of <i>d</i> means separated by a comma: <i>mu</i> <sub>1</sub> , <i>mu</i> <sub>2</sub> , ..., <i>mu</i> <sub><i>d</i></sub> ; <i>A</i> is a positive-definite scale matrix; <i>{var}</i> is a variance parameter
<code>mvn0scaled(d, A, {var})</code>	as <code>mvnscaled()</code> but with zero mean vector



# Built-in Priors (continued)

<code>zellnersg(<i>d</i>,<i>g</i>,<i>mean</i>,{<i>var</i>})</code>	Zellner's <i>g</i> -prior of dimension <i>d</i> with <i>g</i> degrees of freedom, mean vector <i>mean</i> , and variance parameter { <i>var</i> }; <i>mean</i> can be a matrix name or a list of <i>d</i> means separated by comma: <i>mu</i> <sub>1</sub> , <i>mu</i> <sub>2</sub> , ..., <i>mu</i> <sub><i>d</i></sub>
<code>zellnersg0(<i>d</i>,<i>g</i>,{<i>var</i>})</code>	Zellner's <i>g</i> -prior of dimension <i>d</i> with <i>g</i> degrees of freedom, zero mean vector, and variance parameter { <i>var</i> }
<code>dirichlet(<i>a</i><sub>1</sub>,<i>a</i><sub>2</sub>,...,<i>a</i><sub><i>d</i></sub>)</code>	Dirichlet (multivariate beta) of dimension <i>d</i> with shape parameters <i>a</i> <sub>1</sub> , <i>a</i> <sub>2</sub> , ..., <i>a</i> <sub><i>d</i></sub>
<code>wishart(<i>d</i>,<i>df</i>,<i>V</i>)</code>	Wishart of dimension <i>d</i> with degrees of freedom <i>df</i> and scale matrix <i>V</i>
<code>iwishart(<i>d</i>,<i>df</i>,<i>V</i>)</code>	inverse Wishart of dimension <i>d</i> with degrees of freedom <i>df</i> and scale matrix <i>V</i>
<code>jeffreys(<i>d</i>)</code>	Jeffreys prior for covariance of a multivariate normal distribution of dimension <i>d</i>
<code>bernoulli(<i>p</i>)</code>	Bernoulli with success probability <i>p</i>
<code>geometric(<i>p</i>)</code>	geometric for the number of failures before the first success with success probability on one trial <i>p</i>
<code>index(<i>p</i><sub>1</sub>,...,<i>p</i><sub><i>k</i></sub>)</code>	discrete indices 1, 2, ..., <i>k</i> with probabilities <i>p</i> <sub>1</sub> , <i>p</i> <sub>2</sub> , ..., <i>p</i> <sub><i>k</i></sub>
<code>poisson(<i>mu</i>)</code>	Poisson with mean <i>mu</i>
<code>flat</code>	flat prior; equivalent to <code>density(1)</code> or <code>logdensity(0)</code>
<code>density(<i>f</i>)</code>	generic density <i>f</i>
<code>logdensity(<i>logf</i>)</code>	generic log density <i>logf</i>

Dimension *d* is a positive number #.

A distribution argument is a number for scalar arguments such as *var*, *alpha*, *beta*; a Stata matrix for matrix arguments such as *Sigma* and *V*; a model parameter, *paramspec*; an expression, *expr*; or a substitutable expression, *subexpr* or *resubexpr*. See [Specifying arguments of likelihood models and prior distributions](#).

*f* is a nonnegative number, #; an expression, *expr*; or a substitutable expression, *subexpr* or *resubexpr*.

*logf* is a number, #; an expression, *expr*; or a substitutable expression, *subexpr* or *resubexpr*.

When `mvnormal()` or `mvnormal0()` of dimension *d* is applied to *paramref* with *n* parameters (*n* ≠ *d*), *paramref* is reshaped into a matrix with *d* columns, and its rows are treated as independent samples from the specified `mvnormal()` distribution. If such reshaping is not possible, an error is issued. See [example 25](#) for application of this feature.

Where to find it: the [BAYES] bayesmh entry in the Stata PDF manual, under the prior() option. The flat, density(), and logdensity() entries at the bottom are the escape hatches when the catalog does not cover your prior.

# MCMC Controls: Definitions Only

Control	<code>bayesmh</code> option	What it does
<b>Blocking</b>	<code>block()</code>	Groups parameters into joint or separate update steps. Parameters in the same block are proposed together.
<b>Adaptation</b>	<code>adaptation()</code>	Tunes the proposal distribution during burn-in to reach a target acceptance rate.
<b>Thinning</b>	<code>thinning()</code>	Retains every $k$ th draw and, for fixed <code>mcmcsize()</code> , increases the total MH iterations to collect the requested post-burn-in draws.
<b>Burn-in</b>	<code>burnin()</code>	Discards the first $n$ draws before storing.

These settings are out of scope for tuning here. For deterministic replication they still must match across comparator runs.

# Running bayesmh : A First Example

The simplest complete workflow on `auto.dta` : load, fit, and inspect.

```
sysuse auto, clear
quietly replace weight = weight/100

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)
```

- ⊙ `likelihood(normal({var}))` declares the observation model and exposes `{var}` as a parameter you must cover with a prior.
- ⊙ `prior()` covers every declared parameter. Flat on the regression coefficients, Jeffreys on the variance.
- ⊙ `set seed` plus explicit `initial()` pin the starting state so the run is reproducible.

## Reading the Output

```
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: avg = .04811
Log marginal-likelihood = -198.14302 [min, max rows omitted]
```

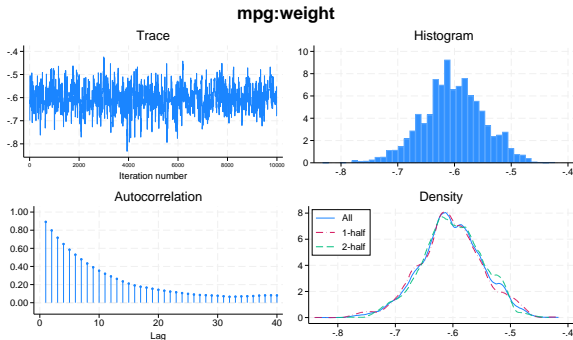
	Mean	Std. dev.	MCSE	Median	[95% cred. int.]	
mpg:weight	-.6026	.0541	.00267	-.6039	-.7115	-.5006
mpg:_cons	39.505	1.6779	.0802	39.455	36.243	43.143
var	12.266	2.1179	.0869	12.053	8.828	17.107

[values rounded; full-precision output in quickstart.log]

- ⦿ **Header.** Sampler, chain sizes, acceptance rate, and efficiency give a first read on how the chain moved.
- ⦿ **Posterior table.** Per parameter: mean, SD, Monte Carlo standard error, median, and 95% equal-tailed credible interval.

# Convergence Diagnostics

```
bayesgraph diagnostics {mpg:weight}
```



## Reading the Four Panels

Four panels, one question each. All four say the same thing on this run: no reason to doubt the posterior.

- ⦿ **Trace.** Is the chain stationary? Look for no drift and no stuck regions. Here: a flat band around  $-0.60$  across 10,000 iterations.
- ⦿ **Histogram.** Is the posterior unimodal? Here: one peak.
- ⦿ **Density.** Does it look the same in the first and second halves of the chain? Here: the *All*, *1-half*, and *2-half* densities overlap.
- ⦿ **Autocorrelation.** Does it decay inside the default lag window? Here: near zero by lag  $\sim 40$ , so successive draws become effectively independent after a short gap.

# bayesstats ess : Effective Sample Size

```
bayesstats ess
```

```
Efficiency summaries      MCMC sample size =    10,000
                          Efficiency:  min =    .04114
                                      avg =    .04811
                                      max =    .05938
```

```
-----
      |      ESS   Corr. time   Efficiency
-----+-----
mpg   |
      |      |
  weight |    411.36      24.31      0.0411
  _cons |    438.19      22.82      0.0438
      |      |
      |      |
  var |    593.75      16.84      0.0594
-----+-----
```

- ⊙ **ESS:** effective number of *independent* draws the chain is worth. About 411 – 594 here, out of 10,000 kept draws.
- ⊙ **Corr. time:** `mcmcsize/ESS`. Lag length needed for draws to be effectively independent; ~17–24 lags on this run.
- ⊙ **Efficiency:** `ESS/mcmcsize`. Fraction of kept draws carrying independent information. About 5% here.

# Scope

## What the rest of this talk covers.

- ⊙ Three ways to specify a likelihood in `bayesmh` : built-in `likelihood()` , inline `llf()` , and user-written `llevuator()` .
- ⊙ A matched-condition replication protocol for validating custom evaluator arithmetic against a built-in comparator.
- ⊙ Worked examples: normal and logistic likelihoods on `auto.dta` in Stata 19.5.

## Assumed background.

- ⊙ Basic Bayesian inference: prior, likelihood, posterior.
- ⊙ MCMC fundamentals: chains, burn-in, acceptance rate, Metropolis–Hastings.
- ⊙ Stata programming: user-written programs, local macros, and positional `args` binding.



# Three Surfaces, One Rule

- ⦿ **Surface 1: Built-in likelihood.**

```
bayesmh mpg weight, likelihood(normal({var})) ...
```

- ⦿ **Surface 2: Inline observation-level log-likelihood.**

```
bayesmh mpg,                                     ///
      likelihood(llf(                             ///
                  lnnormalden(mpg, {mpg:weight}*weight + {mpg:_cons}, ///
                              sqrt({var})))) ...
```

- ⦿ **Surface 3: Custom evaluator program.**

```
bayesmh mpg weight, llevaluator(normalreg, parameters({var})) ...
```

**Between Surface 2 and Surface 3 the syntax changes:** you stop writing the linear predictor yourself and start receiving it as `xb`.

# Stop-Here Boundaries

Surface	Stop here when	What you write / verify
Built-in <code>likelihood()</code>	Stata already supports the model	Built-in command grammar; comparator baseline; no custom-code audit
<code>llf()</code>	Rowwise algebra still fits on one readable line	One substitutable expression; comparator script
<code>llevauator()</code>	Likelihood needs branching logic or extra storage	Program block with observation-level log-likelihood; comparator script

# `llevvaluator()` vs. `evaluator()`

`llevvaluator()` is a deliberately narrow interface. You write only the likelihood; Stata handles priors, proposals, and acceptance.

	<code>llevvaluator()</code>	<code>evaluator()</code>
You write	Observation-level log-likelihood	Log-likelihood and scalar <code>lnprior</code>
Signature	<code>args lnfj xb [params...]</code>	<code>args lnfj lnprior xb [params...]</code>
Stata still handles	Priors, MH, adaptation, diagnostics	MH, adaptation, diagnostics
<code>prior()</code> opt.	Required	Cannot be combined
Sampling	Only Metropolis–Hastings	Only Metropolis–Hastings
Best use	Custom likelihood, standard priors	Prior outside the built-in catalog and must be coded

`evaluator()` is the exception case; `llevvaluator()` remains preferred whenever the prior fits `prior()`.

## Normal Evaluator: Define and Call

```

program normalreg
    version 19.5
    args lnfj xb var
    tempname sd
    scalar 'sd' = sqrt('var')
    quietly replace 'lnfj' = lnnormalden($MH_y, 'xb', 'sd') ///
        if $MH_touse
end

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

```

The whole picture : the evaluator program on top, the `bayesmh` call that invokes it below. The next two slides walk through each block line by line.

# Normal Evaluator: The Full Program

```

program normalreg
  version 19.5
  args lnfj xb var
  tempname sd
  scalar 'sd' = sqrt('var')
  quietly replace 'lnfj' = lnnormalden($MH_y, 'xb', 'sd') ///
    if $MH_touse
end

```

- ⊙ `program normalreg` : defines a Stata program `bayesmh` will call at every MCMC iteration.
- ⊙ `version 19.5` : requests 19.5 syntax/behavior, guarding against future changes.
- ⊙ `args lnfj xb var` : positional binding (Slide 27).
- ⊙ `tempname sd` : temporary scalar name for intermediate storage.
- ⊙ `scalar 'sd' = sqrt('var')` : variance to standard deviation. Getting this wrong is the bug on Slide 33.
- ⊙ `quietly replace 'lnfj' = ...` : the working line (Slide 29). Fills each in-sample row with the observation-level log-likelihood.

## Calling the Evaluator

```

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

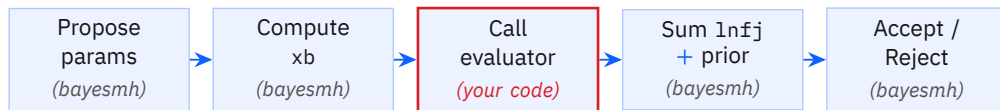
```

Read these two blocks together. The first defines the callback. The second tells `bayesmh` to call it.

`bayesmh` is the host: it allocates the workspace, passes in the inputs, and expects your program to fill the output column correctly. Your evaluator is a guest that fills one column and returns.

## One Evaluator Call, End to End

From the model specification `bayesmh mpg weight`, Stata parses the linear predictor `xb = {mpg:weight}*weight + {mpg:_cons}` and recomputes it at every iteration using the proposal's current coefficient values.



Your evaluator has *one* job: compute the observation-level log-likelihood. It does not propose, it does not accept or reject, and under `llevvaluator()` it does not apply priors.

## dryrun : Ask Stata What It Parsed

The **dryrun** script you run (appendix: `preflight_dryrun.do`):

```
sysuse auto, clear
quietly replace weight = weight/100

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

The parsed-model summary Stata prints back:

```
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.
```



## What `dryrun` Catches

Once you move to `llevaluator()`, the visible summary stops at the callback name:

```
Likelihood:
      mpg ~ normalreg(xb_mpg,{var})
```

Misspell `{var}` as `{vvar}` and `dryrun` catches it: return code `198`, *no prior specified for varname*.

### `dryrun` checks

Likelihood syntax and declarations  
Parameter declarations  
Prior coverage

### `dryrun` does *not* check

Evaluator algebra  
Custom prior algebra inside `evaluator()`  
Posterior propriety

## Priors Are Part of the Model

```
bayesmh mpg weight, likelihood(normal({var})) ///
        prior({mpg:_cons}, normal(25, {var})) ///
        prior({mpg:weight}, normal(0, 100))    ///
        prior({var}, igamma(5, 150))
```

Priors:

```
{mpg:_cons} ~ normal(25,{var})
{mpg:weight} ~ normal(0,100)
{var}       ~ igamma(5,150)
```

- ⦿ Prefer prior specifications that `dryrun` can print back. Priors are part of the model, so comparator runs must match priors exactly.
- ⦿ **Trap:** in `prior(normal())` the second argument is *variance*, not standard deviation. The same confusion shows up inside evaluator code on Slide 33.

# Argument Binding

```
bayesmh mpg weight, llevaluator(normalreg, parameters({var}))
```

What bayesmh provides to the evaluator (by position):

```
Slot 1: lnfj      ← pre-allocated scratch column (fill with log-lik per obs)
Slot 2: xb        ← linear predictor, recomputed each iteration
Slot 3: var       ← first extra parameter from parameters({var})
```

```
program normalreg
  args lnfj  xb  var
      ^    ^   ^
      |    |   +-- binds to slot 3: first extra parameter
      |    +----- binds to slot 2: linear predictor
      +----- binds to slot 1: scratch column
```

`parameters()` binds by position, not by name. Like a patch bay: the first cable goes to the first port.

**There is no runtime type check.** If `parameters()` and `args` are out of order, the wrong value reaches the wrong variable, silently.

# Ownership: What You May Read and Write

`bayesmh` hands your evaluator references into its workspace. Read its inputs. Write only what the contract tells you to fill.

Target	Owner	Your evaluator should	Common mistake
<code>lnfj</code>	<code>bayesmh</code> <code>alloc, you fill</code>	Overwrite every in-sample row; cover every branch	<code>generate</code> on existing column; uncovered branches
<code>xb</code>	<code>bayesmh</code>	Read only	Overwrite it
<code>\$MH_y</code>	<code>bayesmh</code> <code>global</code>	Read the dependent-variable name	Hardcode the outcome
<code>\$MH_touse</code>	<code>bayesmh</code> <code>global</code>	Use as the sample mask on every <code>replace</code>	Let off-sample rows contaminate the sum
Extras from <code>parameters()</code>	<code>bayesmh</code>	Read by position	Assume name-based binding

## The Working Line and `$MH_touse`

```
quietly replace 'lnfj' = lnnormalden($MH_y, 'xb', 'sd') if $MH_touse
```

- ⦿ **By contract, `$MH_touse` is mandatory.**
- ⦿ **`replace`, not `generate`** . The scratch column already exists; `generate` would fail on iteration two.
- ⦿ **`quietly` suppresses per-call output.** Without it, Stata prints a substitution message on every evaluation.

## How the `$MH_touse` Guard Works

Row	mpg	<code>\$MH_touse</code>	Action under <code>replace ... if \$MH_touse</code>
1 (in sample)	22	1	Overwrites <code>lnfj</code> with new log-likelihood
2 (missing data)	.	0	Ignores row; <code>lnfj</code> keeps scratch value
3 (in sample)	17	1	Overwrites <code>lnfj</code> with new log-likelihood

Your job is to fill the in-sample cells, not to build the sheet from scratch.

# Correctness Bugs in Custom Evaluators

Bug	Consequence
Pass variance to <code>lnnormalden()</code> instead of SD	Wrong posterior: code runs, density is wrong, posterior shifts
Omit <code>if \$MH_touse</code>	Wrong posterior: off-sample rows contribute to the total log-likelihood
Swap <code>parameters()</code> order	Wrong posterior: wrong holder bound to wrong local macro; runs silently
Leave a branch uncovered	Wrong posterior: stale <code>lnfj</code> from previous proposal persists
Hardcode the outcome name	Fragile: evaluator only works for one dataset layout

The first four mistakes silently shift the posterior; the last makes the evaluator nonportable. Matched-condition comparison catches the first four when a built-in comparator exists.

## Correct vs. Wrong

```
// CORRECT
scalar 'sd' = sqrt('var')
quietly replace 'lnfj' = lnnormalden($MH_y, 'xb', 'sd')      ///
    if $MH_touse

// WRONG: passing variance, not standard deviation
quietly replace 'lnfj' = lnnormalden($MH_y, 'xb', 'var')    ///
    if $MH_touse

// WRONG: uncovered branch (only covers y == 1)
quietly replace 'lnfj' = ln(invlogit('xb'))                ///
    if $MH_y == 1 & $MH_touse
// If y == 0, lnfj is unchanged and retains
// stale values from the previous iteration.
```

For evaluators with outcome-conditional branches, assert the data domain before estimation:

```
assert inlist(foreign, 0, 1) if !missing(foreign, mpg)
```



## Catching the Canonical Bug

The bug is simple: pass variance instead of SD to `lnnormalden()`.  
Matched-condition comparison against a deliberately broken evaluator:

Metric	Built-in	Broken evaluator	Absolute diff
weight mean	-0.6025616	-0.6020815	4.80e-4
_cons mean	39.50491	39.48684	1.81e-2
var mean	12.26586	3.47327	8.79259
Acceptance rate	.1668	.2504	.08359
e(lml_lm)	-198.14302	-198.93781	.79479

Same likelihood, priors, MCMC, seed, initials, data, Stata version, but different posterior. 3.47327 is not random; it is near  $\sqrt{12.26586} \approx 3.50$  because the broken `{var}` is forced into the SD slot.

**If you had compared only the regression coefficients, you might have missed the bug entirely.**

# When Missing Values Lie

## Boundary rejection

A proposal steps outside the legal domain,  
e.g. `sqrt(var)` with `var < 0`

Missing `lnfj` rejects the proposal

*Acceptable and expected*

The sampler has no way to distinguish “the proposal landed outside the domain” from “the formula has a bug.”

## Broken arithmetic

The formula itself returns missing for a legal proposal

Missing `lnfj` also rejects the proposal

*Evaluator is broken*

## Broken Logistic Evaluator

Broken evaluator used in the failure-mode demonstration:

```

program logitmiss
  version 19.5
  args lnfj xb
  quietly replace 'lnfj' = .                ///
                if abs('xb') > 2 & $MH_touse
  quietly replace 'lnfj' = ln(invlogit('xb'))  ///
                if abs('xb') <= 2 & $MH_y == 1 & $MH_touse
  quietly replace 'lnfj' = ln(invlogit(-'xb'))  ///
                if abs('xb') <= 2 & $MH_y == 0 & $MH_touse
end

```

This is a failure-mode demonstration, not a comparator run. It uses `mcmcsz(500) burnin(200)` and does not match the baseline MCMC settings. Its point: the log reaches a model summary instead of stopping with an error, because missing `lnfj` is interpreted as rejection. Bad arithmetic can finish, reject proposals, and still corrupt inference.

# Matched-Condition Replication Protocol

- 1 **Baseline.** Fit the model with the built-in likelihood under a fixed seed, explicit initial values, and declared priors.
- 2 **Test.** Write the custom evaluator. Fit the same model with `l1evaluator()` under *identical* matched conditions: same likelihood, priors, MCMC settings, parameterization, data, seed, initial values, and Stata version.
- 3 **Extract.** Store summaries from both runs: means, SDs, MCSEs, medians, credible intervals, acceptance rate.
- 4 **Compare.** Compute absolute differences across all extracted metrics.

# Logistic Baseline

Built-in comparator:

```
set seed 14
bayesmh foreign mpg, likelihood(logit)          ///
      prior({foreign:}, flat) initial({foreign:} 0)
```

The normal family has an unknown variance that must be estimated, requiring `parameters({var})`. The binomial family has a fixed dispersion parameter of 1, so all model uncertainty is captured by the regression coefficients in `xb`. The logistic evaluator has no `parameters()` declaration; its signature is just `args lnfj xb`.

**Three failure modes in double-precision arithmetic.** Double-precision floating point can represent values very close to 1 but not infinitely close. There are only finitely many representable values near 1, and `invlogit(37)` crosses that boundary and rounds all the way to `1.0`.

## Three Arithmetic Failure Boundaries

Case	Expression	Failure mode	Threshold
$y = 0$	<code>ln(1 - invlogit(xb))</code>	Cancellation after <code>invlogit(xb)</code> rounds to 1	$xb \approx 37$
$y = 0$	<code>ln(invlogit(-xb))</code>	<code>exp(xb)</code> overflows inside <code>invlogit(-xb)</code>	$xb \approx 709.78$
$y = 1$	<code>ln(invlogit(xb))</code>	<code>exp(-xb)</code> overflows; <code>invlogit(xb) = 0</code> ; <code>ln(0)</code> missing	$xb \approx -709.78$

The manual's shorter `ln(invlogit())` form matches the built-in on `auto.dta`. For datasets with more extreme predictors, the piecewise rewrite on the next slide avoids all three failure boundaries.

## Safe Logistic Evaluator: Derivation

To avoid evaluating `invlogit()` directly, the log-probability decomposes into branches. For `y = 1` (i.e. `log(sigmoid(xb))`):

<code>log(sigmoid(xb)) = -ln1p(exp(-xb))</code>	for <code>xb &gt;= 0</code>
<code>log(sigmoid(xb)) = xb - ln1p(exp(xb))</code>	for <code>xb &lt; 0</code>

For `y = 0`, the complementary identity `1 - sigmoid(xb) = sigmoid(-xb)` converts the problem to the same functional form, then expands identically:

<code>log(sigmoid(-xb)) = -xb - ln1p(exp(-xb))</code>	for <code>xb &gt;= 0</code>
<code>log(sigmoid(-xb)) = -ln1p(exp(xb))</code>	for <code>xb &lt; 0</code>

# logitllsafe

```

program logitllsafe
  version 19.5
  args lnfj xb
  // y = 1 branches
  quietly replace 'lnfj' = -ln1p(exp(-'xb'))          ///
    if $MH_y == 1 & 'xb' >= 0 & $MH_touse
  quietly replace 'lnfj' = 'xb' - ln1p(exp('xb'))      ///
    if $MH_y == 1 & 'xb' < 0 & $MH_touse
  // y = 0 branches (via 1 - sigmoid(xb) = sigmoid(-xb))
  quietly replace 'lnfj' = -'xb' - ln1p(exp(-'xb'))    ///
    if $MH_y == 0 & 'xb' >= 0 & $MH_touse
  quietly replace 'lnfj' = -ln1p(exp('xb'))            ///
    if $MH_y == 0 & 'xb' < 0 & $MH_touse
end

```

```

bayesmh foreign mpg, llevaluator(logitllsafe) ///
  prior({foreign:}, flat) initial({foreign:} 0)

```

When `invlogit()` saturates, `ln()` of the saturated value is either trivially zero or undefined. The piecewise rewrite avoids `invlogit()` entirely, so no saturation can occur.



## Logistic Validation Result

Both logistic evaluators match the built-in within machine precision. With MCSE around  $1.67\text{e-}3$  on these parameters, the largest drift ( $\sim 1\text{e-}14$ ) is about eleven orders of magnitude smaller.

Evaluator		Max summary diff*	Acceptance-rate diff	e(1ml_1m)	diff†
Manual	<code>ln(invlogit())</code>	$\sim 3.6\text{e-}14$	0	$\sim 7.1\text{e-}15$	
Piecewise	<code>ln1p()</code>	$\sim 2.4\text{e-}14$	0	$\sim 1.4\text{e-}14$	

\*Max absolute difference across mean, SD, MCSE, median, credible interval.

† Optional supplementary check.

Identical arithmetic paths give exact identity; rewritten paths give machine-precision drift.

Local evidence on `auto.dta` (scope: Slide 16). This validates normal and logistic only. The next evaluator you write still needs its own comparator run.

## Practice: The Blueprint Library ( [R] glm )

### Your turn

Pick a GLM family from [R] glm Methods and Formulas and carry the sketch below through the five-step protocol from Slide 36. The blueprint tells you which pieces stay fixed and which you have to rewrite.

#### What stays fixed

Evaluator contract: `args lnfj xb ... , fill lnfj , guard with $MH_touse`

#### What changes by family

Inverse link and observation-level log-likelihood formula

For families with ancillary parameters (gamma, inverse Gaussian, negative binomial), [R] glm gives the likelihood kernel; you still must choose a Bayesian parameterization and prior for those extras in `bayesmh` .

Starter sketch, Poisson with log link (*not* yet validated):

```
// For the log link, mu = exp(xb) substitutes directly.
// Same evaluator contract as normalreg.
quietly replace 'lnfj' = $MH_y*'xb' - exp('xb') - ///
    lngamma($MH_y + 1) if $MH_touse
```

## Weaker Checks (Part 1)

**The protocol ends here. What follows are engineering heuristics for cases where the protocol cannot be applied.**

When no built-in comparator exists, work down a staged cross-check list. Treat a novel evaluator like new lab hardware: calibrate the piece you can measure directly, then add one moving part at a time.

- 1 **Simplified subset.** If your evaluator handles a complex model (e.g., zero-inflated Poisson), validate the simpler component alone first against a built-in comparator (e.g., Poisson against `likelihood(poisson)` ), then add one moving part at a time.
- 2 **Fixed-parameter spot checks.** Evaluate the log-likelihood at chosen parameter values and compare with an independent calculation.
- 3 **Simulated-data recovery.**
  - ▷ *Calibration*: generate repeated toy datasets from known parameters. Do credible intervals cover the true values at the expected rate? (Stronger.)
  - ▷ *Sanity*: on a single toy dataset, do posterior centers and intervals land near the generating value? (Necessary, not sufficient.)

## Weaker Checks (Part 2)

- 4 **Tiny-data hand calculation.** Use a dataset small enough to compute the rowwise contributions yourself.
- 5 **Independent reimplementations.** Recompute the same rowwise likelihood in another code path and compare observation by observation.

When the comparator disappears, the evidence gets weaker. *It does not become optional.*

# Summary

- ⦿ Start with the smallest `bayesmh` surface that fits the model.
- ⦿ Run `dryrun` before every chain. It checks parsing and prior coverage, not evaluator algebra.
- ⦿ Read the evaluator as a contract: under `llevauator()`, `bayesmh` owns proposals, `xb`, priors, and acceptance; your code owns `lnfj` and the sample guard.
- ⦿ Positional mistakes are silent because there is no runtime type check.
- ⦿ Validate custom evaluator code by matched-condition replication against a built-in comparator whenever one exists: same likelihood, priors, MCMC settings, parameterization, data, seed, initial values, and Stata version.
- ⦿ **Validation standard:** exact zero when the arithmetic path is identical; example-justified machine-precision tolerance when the algebra is rewritten. `normalreg` gives exact zero (same `lnnormalden()` path); the logistic evaluator gives  $\sim 1e-14$  drift (rewritten `ln1p()` path). Normal and logistic only.
- ⦿ If no comparator exists, say plainly that the evidence is weaker and work down the staged cross-check list.

**Before trusting a custom evaluator: run `dryrun`, match the model and sampler, fix seed and initials, and compare against a built-in baseline.**

# References

- 1 StataCorp. *Stata 19 Bayesian Analysis Reference Manual*. [BAYES] bayesmh : Likelihood-model specification, Prior specification, Checking model specification, MCMC sampling options, Remarks on initial values, Methods and Formulas.
- 2 StataCorp. *Stata 19 Bayesian Analysis Reference Manual*. [BAYES] bayesmh evaluators : Syntax, Remarks, Examples, Prediction, Mata-based evaluators.
- 3 StataCorp. *Stata 19 Bayesian Analysis Reference Manual*. [BAYES] bayes : Remarks on bayes: versus bayesmh .
- 4 StataCorp. *Stata 19 Base Reference Manual*. [FN] Statistical functions : lnnormalden(x, mu, sd) .
- 5 StataCorp. *Stata 19 Base Reference Manual*. [R] glm : Methods and Formulas, for family-specific log-likelihood formulas.
- 6 StataCorp. *Stata 19 Programming Reference Manual*. [P] version .
- 7 StataCorp. *Stata 19 Programming Reference Manual*. [P] macro : Local macros and backtick dereferencing syntax.
- 8 StataCorp. *Stata 19 Programming Reference Manual*. [P] syntax : args command for positional argument binding.