Fitting Models by Bayesian Inference with JAGS

Last modified: 2026-11-12 22:02:32 (PST)

This page fits models by Bayesian inference, using the JAGS sampler driven from R: a single proportion, a logistic regression, a survival model, and a random-effects model, and then averages over linear regression models [1, Ch. 14]. After the first example, the page compares JAGS with Stan on the same model. It uses the priors of the Bayesian Inference page and the sampling and convergence checks of the Markov Chain Monte Carlo page.

A first example: a single proportion

From here on, the models are fit by Bayesian inference with JAGS ("Just Another Gibbs Sampler"), a program that builds an MCMC sampler from a text description of a model [1, Ch. 13], driven from R through the `rjags` package. We begin with the simplest possible model, a single Bernoulli probability, whose exact posterior is known from the Beta-Bernoulli example, so the output can be checked. The example shows the mechanics that every later analysis reuses: specifying a model, supplying data, running a burn-in, monitoring parameters, and summarizing and checking the draws. In JAGS, `dnorm(mean, precision)` is parameterized by the precision, the reciprocal of the variance.

Example 0.1 (A Bernoulli model fitted with JAGS)**.** The data are $r = 55$ successes in $n = 91$ trials, and the prior is uniform, as in the Beta-Bernoulli example. Each chain gets its own random-number generator seed, so the output is reproducible:

```r
y <- rep(c(1L, 0L), times = c(55L, 36L))
bernoulli_model <- "
model {
  for (i in 1:N) {
    y[i] ~ dbern(p)
  }
  p ~ dbeta(1, 1)
}
"
inits <- list(
  list(.RNG.name = "base::Mersenne-Twister", .RNG.seed = 1),
  list(.RNG.name = "base::Mersenne-Twister", .RNG.seed = 2)
)
fit <- rjags::jags.model(
  textConnection(bernoulli_model),
  data = list(y = y, N = length(y)),
  inits = inits,
  n.chains = 2,
  quiet = TRUE
)
stats::update(fit, n.iter = 1000, progress.bar = "none") # burn-in
draws <- rjags::coda.samples(
  fit,
  variable.names = "p",
  n.iter = 4000,
  progress.bar = "none"
)
p_draws <- as.matrix(draws)[, "p"]
c(
  mean = mean(p_draws),
  stats::quantile(p_draws, c(0.025, 0.975)),
  Rhat = coda::gelman.diag(draws)$psrf[1, "Point est."]
) |>
  round(3)
#>  mean  2.5% 97.5%  Rhat
#> 0.601 0.499 0.698 1.000
```

The posterior mean and equal-tailed 95% credible interval agree with the exact Beta$(56, 37)$ values of the Beta-Bernoulli example to about two decimal places, and the potential scale reduction factor is 1.

Comparing Stan and JAGS

JAGS is not the only program that turns a model description into an MCMC sampler. Stan [2] is a widely used alternative. Both take a model written in their own language, accept data from R, and return draws from the posterior. They differ in how they draw and in what a model may contain.

JAGS updates one node of the model at a time, in the manner of a Gibbs sampler [3], [4]. For each full conditional, it uses a conjugate sampler where it recognizes one, and slice or Metropolis steps otherwise. Stan uses Hamiltonian Monte Carlo, which proposes moves along the gradient of the logarithm of the posterior, in the adaptive form called the No-U-Turn sampler (NUTS) [5]. Using the gradient is why Stan cannot draw a discrete parameter: a discrete parameter has no gradient. A Stan model sums a discrete quantity out of the joint density of the data and the parameters instead [6], chapter "Latent Discrete Parameters". Table 1 lists the practical differences, taken from the JAGS manual [4] and the Stan User's Guide [6]:

Table 1: JAGS and Stan compared

Feature	JAGS	Stan
Sampling algorithm	Updates one node at a time, as in Gibbs sampling, with conjugate samplers where it recognizes them and slice or Metropolis steps otherwise	Hamiltonian Monte Carlo, in its No-U-Turn form (NUTS)
Model language	A BUGS-style description of a graphical model; the order of the statements does not matter	A typed program in blocks (`data`, `parameters`, `model`); the statements run in order and accumulate the log density
Normal distribution	`dnorm(mean, precision)`	`normal(mean, sd)`
Discrete parameters	Allowed; JAGS samples them	Not allowed; they are summed out of the joint density
Execution	Interpreted by the JAGS library	Translated to C++ and compiled before sampling

Table 1: JAGS and Stan compared

Feature	JAGS	Stan
Convergence checks	`Rhat` and effective sample size, computed from the draws with the `coda` package	`Rhat` and effective sample size, plus sampler checks such as divergent transitions and tree depth
R interfaces	`rjags`, `runjags`	`rstan`, `cmdstanr`
Official site	mcmc-jags.sourceforge.io	mc-stan.org

The next example fits the model of the first example in Stan, so the two programs can be compared on one problem.

Example 0.1 (The Bernoulli model fitted with Stan)**.** The data and the uniform prior are those of the JAGS example. The Stan program declares the data and the parameter with their types and ranges, and the `model` block states the prior and the likelihood:

```
bernoulli_stan <- "
data {
  int<lower=0> N;
  array[N] int<lower=0, upper=1> y;
}
parameters {
  real<lower=0, upper=1> p;
}
model {
  p ~ beta(1, 1);
  y ~ bernoulli(p);
}
"
stan_fit <- rstan::stan(
  model_code = bernoulli_stan,
  data = list(N = length(y), y = y),
  chains = 2,
  warmup = 1000,
  iter = 5000,
  seed = 1,
  refresh = 0
)
stan_draws <- as.matrix(stan_fit, pars = "p")[, "p"]
stan_summary <- rstan::summary(stan_fit, pars = "p")$summary
round(stan_summary[, c("mean", "2.5%", "97.5%", "Rhat")], 3)
#>  mean  2.5% 97.5%  Rhat
#> 0.602 0.501 0.698 1.000
```

Table 2 puts the two programs beside the posterior of the Beta-Bernoulli example, whose Beta parameters come from the number of successes and failures:

Table 2: Posterior of the proportion from JAGS, Stan and the Beta posterior

	Mean	2.5%	97.5%
JAGS	0.601	0.499	0.698
Stan	0.602	0.501	0.698
Beta	0.602	0.501	0.699

Stan's split potential scale reduction factor is 1, and the sampler reports 0 divergent transitions. As Table 2 shows, the mean and the interval endpoints from each program differ from those of

the Beta posterior by at most 0.002. The programs differ in the model code: the Stan program carries the types and ranges of its variables, and it is compiled before it draws.

Binary outcomes: logistic regression

For a binary outcome $Y_i \sim \text{Bernoulli}(\pi_i)$ with $\text{logit}(\pi_i) = x_i \cdot \beta$, the logistic regression model, a Bayesian analysis places a prior on the coefficient vector β and samples the posterior $\mathbb{p}(\beta \mid y)$ by MCMC [1, Ch. 14], p. 318. Each β_{x_j} is summarized by the mean and quantiles of its draws. No large-sample Gaussian approximation is needed: a credible interval is read directly from the posterior quantiles, and the posterior of an odds ratio $e^{\beta_{x_j}}$, or of any other function of β, is obtained by transforming the draws. For a video introduction, see Richard McElreath's lecture *Modeling Events* (Statistical Rethinking 2026, Lecture A09).

Example 0.1 (Logistic regression fitted by Bayesian inference)**.** We simulate $N = 200$ observations from a logistic model with one continuous predictor, intercept $\beta_0 = -0.5$ and slope $\beta_x = 1.2$, and place a diffuse $\text{N}(0, 10^2)$ prior on each coefficient (precision 0.01 in JAGS). The JAGS program also monitors the odds ratio e^{β_x} for a one-unit increase in x. In the code, `beta0` and `beta1` are β_0 and β_x.

```r
set.seed(2024)
n_obs <- 200L
x <- stats::rnorm(n_obs)
y <- stats::rbinom(n_obs, size = 1, prob = stats::plogis(-0.5 + 1.2 * x))

logistic_model <- "
model {
  for (i in 1:N) {
    y[i] ~ dbern(pi[i])
    logit(pi[i]) <- beta0 + beta1 * x[i]
  }
  beta0 ~ dnorm(0, 0.01)
  beta1 ~ dnorm(0, 0.01)
  OR <- exp(beta1)
}
"
inits <- list(
  list(.RNG.name = "base::Mersenne-Twister", .RNG.seed = 1),
  list(.RNG.name = "base::Mersenne-Twister", .RNG.seed = 2)
)
fit <- rjags::jags.model(
  textConnection(logistic_model),
  data = list(y = y, x = x, N = n_obs),
  inits = inits,
  n.chains = 2,
  quiet = TRUE
)
stats::update(fit, n.iter = 1000, progress.bar = "none")
draws <- rjags::coda.samples(
  fit,
  variable.names = c("beta0", "beta1", "OR"),
  n.iter = 4000,
  progress.bar = "none"
)
draws_matrix <- as.matrix(draws)
cbind(
  mean = colMeans(draws_matrix),
  t(apply(draws_matrix, 2, stats::quantile, probs = c(0.025, 0.975))),
  Rhat = coda::gelman.diag(draws, multivariate = FALSE)$psrf[, "Point est."]
) |>
  round(3)
#>         mean   2.5%  97.5%  Rhat
#> OR     3.417  2.275  5.132 1.000
#> beta0 -0.654 -1.002 -0.316 1.001
#> beta1  1.207  0.822  1.635 1.000
```

Each 95% credible interval contains its coefficient's true value, and every $\hat{R}$ is near 1. As comparing MCMC estimates to maximum likelihood leads us to expect, the posterior means are close to the maximum likelihood estimates:

```
stats::glm(y ~ x, family = stats::binomial()) |>
  stats::coef() |>
  round(3)
#> (Intercept)           x
#>      -0.647       1.183
```

Survival analysis

Parametric survival models, such as those with exponential or Weibull event times, can be fit by Bayesian inference: priors are placed on the baseline-hazard parameters and the regression coefficients, and the posterior is sampled by MCMC [1, Ch. 14], p. 327. Censoring enters through the likelihood, exactly as in maximum likelihood estimation with the likelihood with censoring: an observed event at time t contributes its density $\lambda(t)\,\mathbb{S}(t)$, and an observation right-censored at time t contributes its survival probability $\mathbb{S}(t)$, where λ is the hazard and $\mathbb{S}$ the survival function.

When a model's log-likelihood contribution ℓ_i for observation i is not one of the distributions built into JAGS, the *zeros trick* supplies it. An observed value of 0 from a Poisson distribution with mean ϕ_i has probability $e^{-\phi_i}$, and with $\phi_i \overset{\text{def}}{=} C - \ell_i$ for a constant C,

$$\begin{aligned} e^{-\phi_i} &= e^{-(C-\ell_i)} && \text{(definition of } \phi_i\text{)} \\ &= e^{-C} e^{\ell_i} && \text{(splitting the exponent)} \\ &\propto e^{\ell_i} && (C \text{ does not depend on the parameters}), \end{aligned}$$

which is observation i's likelihood contribution. So declaring data `zeros[i] = 0` with `zeros[i] ~ dpois(C - loglik[i])` multiplies the likelihood by e^{ℓ_i}. The constant C only has to be large enough to keep every ϕ_i positive.

Example 0.1 (Exponential survival regression fitted by Bayesian inference)**.** We simulate $N = 200$ right-censored exponential survival times, with a binary covariate x_i (say, treatment) and hazard $\lambda_i = \exp\{\beta_0 + \beta_x x_i\}$, where the log baseline hazard is $\beta_0 = \log 0.05$ and the log hazard ratio is $\beta_x = -0.7$. With event indicator δ_i (1 for an event, 0 for a censored time) and follow-up time t_i, the exponential density is $\lambda_i e^{-\lambda_i t}$ and its survival function is $e^{-\lambda_i t}$, so observation i contributes the log-likelihood

$$\ell_i \stackrel{\text{def}}{=} \delta_i \log \lambda_i - \lambda_i t_i.$$

JAGS has no built-in distribution for this contribution, so we fit it with the zeros trick. In the code, `beta0` and `beta1` are β_0 and β_x.

```r
set.seed(2026)
n_obs <- 200L
x <- stats::rbinom(n_obs, size = 1, prob = 0.5)
event_time <- stats::rexp(n_obs, rate = exp(log(0.05) - 0.7 * x))
censor_time <- stats::rexp(n_obs, rate = 0.03)
follow_up <- pmin(event_time, censor_time)
event <- as.integer(event_time <= censor_time)

survival_model <- "
model {
  C <- 10000
  for (i in 1:N) {
    log(lambda[i]) <- beta0 + beta1 * x[i]
    loglik[i] <- event[i] * log(lambda[i]) - lambda[i] * follow_up[i]
    zeros[i] ~ dpois(C - loglik[i])
  }
  beta0 ~ dnorm(0, 0.01)
  beta1 ~ dnorm(0, 0.01)
  HR <- exp(beta1)
}
"
inits <- list(
  list(.RNG.name = "base::Mersenne-Twister", .RNG.seed = 1),
  list(.RNG.name = "base::Mersenne-Twister", .RNG.seed = 2)
)
fit <- rjags::jags.model(
  textConnection(survival_model),
  data = list(
    follow_up = follow_up, x = x, event = event,
    N = n_obs, zeros = rep(0, n_obs)
  ),
  inits = inits,
  n.chains = 2,
  quiet = TRUE
)
stats::update(fit, n.iter = 1000, progress.bar = "none")
draws <- rjags::coda.samples(
  fit,
  variable.names = c("beta0", "beta1", "HR"),
  n.iter = 4000,
  progress.bar = "none"
)
draws_matrix <- as.matrix(draws)
cbind(
  mean = colMeans(draws_matrix),
  t(apply(draws_matrix, 2, stats::quantile, probs = c(0.025, 0.975))),
  Rhat = coda::gelman.diag(draws, multivariate = FALSE)$psrf[, "Point est."]
) |>
  round(3)
#>         mean   2.5%  97.5% Rhat
#> HR     0.602  0.390  0.875    1
#> beta0 -3.119 -3.391 -2.870    1
#> beta1 -0.509 -0.941 -0.133    1
```

About 50% of the follow-up times end in an event. The 95% credible interval for β_x contains the true -0.7, though the posterior mean is some distance from it. The maximum likelihood estimates, from the same log-likelihood, show that the gap is sampling variation in this data set rather than an effect of the prior:

```
neg_loglik <- function(beta) {
  lambda <- exp(beta[1] + beta[2] * x)
  -sum(event * log(lambda) - lambda * follow_up)
}
stats::optim(c(0, 0), neg_loglik)$par |>
  stats::setNames(c("beta0", "beta1")) |>
  round(3)
#>  beta0  beta1
#> -3.112 -0.524
```

The monitored e^{β_x} gives the hazard ratio and its credible interval directly.

Random effects

The hierarchical model of the two-level Gaussian example is the random-effects *model structure*, with group-level parameters $\theta_j \sim \mathrm{N}(\mu, \tau^2)$ drawn from a common distribution [7]. What changes here from a maximum likelihood fit is the *inference method*: the Bayesian analysis places priors on the hyperparameters μ and τ and samples their joint posterior together with the group-level parameters [1, Ch. 14], p. 329. The posterior shrinks each group's estimate toward the overall mean, by an amount the data determine through τ, and the posterior for τ carries the uncertainty about the between-group spread into every group-level summary.

Example 0.1 (A random-intercept model fitted by Bayesian inference)**.** We simulate $J = 8$ groups of 12 observations each, with group means drawn from $\mathrm{N}(\mu, \tau^2)$ ($\mu = 5$, $\tau = 1.5$) and within-group standard deviation $\sigma = 2$. The priors are a diffuse $\mathrm{N}(0, 100^2)$ for μ and flat priors on $(0, 100)$ for the standard deviations τ and σ. Being flat on the standard-deviation scale is a choice: as a flat prior on the log-odds shows for a probability, it is not flat on another scale, such as the variance.

```r
set.seed(2025)
n_groups <- 8L
n_per_group <- 12L
theta_true <- stats::rnorm(n_groups, mean = 5, sd = 1.5)
group <- rep(seq_len(n_groups), each = n_per_group)
y <- stats::rnorm(n_groups * n_per_group, mean = theta_true[group], sd = 2)

random_intercept_model <- "
model {
  for (i in 1:N) {
    y[i] ~ dnorm(theta[group[i]], 1 / sigma^2)
  }
  for (j in 1:J) {
    theta[j] ~ dnorm(mu, 1 / tau^2)
  }
  mu ~ dnorm(0, 1.0E-4)
  sigma ~ dunif(0, 100)
  tau ~ dunif(0, 100)
}
"
inits <- list(
  list(.RNG.name = "base::Mersenne-Twister", .RNG.seed = 1),
  list(.RNG.name = "base::Mersenne-Twister", .RNG.seed = 2)
)
fit <- rjags::jags.model(
  textConnection(random_intercept_model),
  data = list(y = y, group = group, N = length(y), J = n_groups),
  inits = inits,
  n.chains = 2,
  quiet = TRUE
)
stats::update(fit, n.iter = 2000, progress.bar = "none")
draws <- rjags::coda.samples(
  fit,
  variable.names = c("mu", "tau", "sigma", "theta"),
  n.iter = 10000,
  progress.bar = "none"
)
draws_matrix <- as.matrix(draws)
posterior_summary <- cbind(
  mean = colMeans(draws_matrix),
  t(apply(draws_matrix, 2, stats::quantile, probs = c(0.025, 0.975))),
  Rhat = coda::gelman.diag(draws, multivariate = FALSE)$psrf[, "Point est."]
)
round(posterior_summary[c("mu", "tau", "sigma"), ], 3)
#>        mean  2.5% 97.5%  Rhat
#> mu    5.572 4.733 6.405 1.002
#> tau   0.868 0.125 2.002 1.004
#> sigma 2.161 1.867 2.520 1.000
```

The 95% credible intervals contain the true $\mu = 5$, $\tau = 1.5$ and $\sigma = 2$. The interval for τ is wide: eight groups say little about how spread out group means are, and the posterior reports that uncertainty instead of a single estimate of the variance component.
Each group's posterior mean lies between its sample mean and the overall sample mean:

```
sample_means <- as.vector(tapply(y, group, mean))
tibble::tibble(
  group_id = seq_len(n_groups),
  sample_mean = sample_means,
  posterior_mean = posterior_summary[
    paste0("theta[", seq_len(n_groups), "]"), "mean"
  ]
) |>
  dplyr::mutate(overall_mean = mean(y)) |>
  round(2)
#> # A tibble: 8 x 4
#>   group_id sample_mean posterior_mean overall_mean
#>      <dbl>       <dbl>          <dbl>        <dbl>
#> 1        1        6.08           5.87         5.56
#> 2        2        5.53           5.55         5.56
#> 3        3        6.14           5.9          5.56
#> 4        4        7.27           6.57         5.56
#> 5        5        5.1            5.3          5.56
#> 6        6        4.31           4.85         5.56
#> 7        7        5.51           5.53         5.56
#> 8        8        4.56           4.99         5.56
```

This is the borrowing of strength described in distributions and hierarchies: group 4, with the largest sample mean, is pulled furthest toward the overall mean.

Bayesian model averaging

When several candidate models are plausible, committing to a single "best" one ignores the uncertainty about which model is right. Bayesian model averaging instead averages over the models, weighting each by its posterior probability [1, Ch. 14], p. 339. This approach carries *model* uncertainty, not just parameter uncertainty, into the final inference. It is an alternative to choosing a single model by predictor selection.

Definition 0.1 (Posterior model probability)**.** Let $M_1, \dots, M_K$ be candidate models with prior probabilities $\mathbb{p}(M_k)$ summing to 1, and let $\mathbb{p}(\underset{\sim}{y} \mid M_k)$ be the marginal likelihood of the data under model M_k. The **posterior model probability** of M_k is

$$\mathbb{p}(M_k \mid \underset{\sim}{y}) \stackrel{\text{def}}{=} \frac{\mathbb{p}(\underset{\sim}{y} \mid M_k)\,\mathbb{p}(M_k)}{\sum_{l=1}^{K} \mathbb{p}(\underset{\sim}{y} \mid M_l)\,\mathbb{p}(M_l)}.$$

Example 0.1 (Posterior probabilities of two models)**.** Two models with equal prior probabilities

$$\begin{aligned}\mathbb{p}(M_1) &= \mathbb{p}(M_2)\\ &= 0.5\end{aligned}$$

and marginal likelihoods $\mathbb{p}(\underset{\sim}{y} \mid M_1) = 0.02$ and $\mathbb{p}(\underset{\sim}{y} \mid M_2) = 0.06$ have posterior probabilities

$$\begin{aligned}\mathbb{p}(M_1 \mid \underset{\sim}{y}) &= 0.01/(0.01 + 0.03)\\ &= 0.25\end{aligned}$$

and $\mathbb{p}(M_2 \mid \underset{\sim}{y}) = 0.75$.

Definition 0.2 (Bayesian model averaging)**.** With the posterior model probabilities $\mathbb{p}(M_k \mid \underset{\sim}{y})$ of candidate models $M_1, \dots, M_K$, **Bayesian model averaging** estimates a quantity Δ that has the same meaning in every model by its posterior distribution averaged over the models:

$$\mathbb{p}(\Delta \mid \underset{\sim}{y}) \stackrel{\text{def}}{=} \sum_{k=1}^{K} \mathbb{p}(\Delta \mid M_k, \underset{\sim}{y})\,\mathbb{p}(M_k \mid \underset{\sim}{y}).$$

Definition 0.3 (Posterior inclusion probability)**.** When the candidate models $M_1, \dots, M_K$ of Definition 0.2 differ in which predictors they include, the **posterior inclusion probability** of a predictor x_j is the sum of the posterior model probabilities of the models that include it:

$$\text{PIP}_j \stackrel{\text{def}}{=} \sum_{k:\, x_j \in M_k} \mathbb{p}(M_k \mid \underset{\sim}{y}).$$

Example 0.2 (A BIC approximation to Bayesian model averaging)**.** Marginal likelihoods are hard to compute, but with equal prior probabilities for the models, the Bayesian information criterion gives a large-sample approximation to the posterior model probabilities

[8] [9]:

$$\mathbb{p}(M_k \mid \underset{\sim}{y}) \approx \frac{\exp\{-\frac{1}{2}\,\mathrm{BIC}_k\}}{\sum_{l=1}^{K} \exp\{-\frac{1}{2}\,\mathrm{BIC}_l\}}.$$

We simulate data in which only x_1 and x_2 affect the outcome, fit a linear regression for every subset of the three predictors x_1, x_2 and x_3, and compute these approximate posterior model probabilities:

```r
set.seed(2027)
n_obs <- 120L
dat <- tibble::tibble(
  x1 = stats::rnorm(n_obs),
  x2 = stats::rnorm(n_obs),
  x3 = stats::rnorm(n_obs)
) |>
  dplyr::mutate(y = 1 + 0.8 * x1 + 0.5 * x2 + stats::rnorm(n_obs))

predictors <- c("x1", "x2", "x3")
subsets <- lapply(0:3, function(k) {
  utils::combn(predictors, k, simplify = FALSE)
}) |>
  unlist(recursive = FALSE)
models <- lapply(subsets, function(vars) {
  rhs <- if (length(vars) > 0) paste(vars, collapse = " + ") else "1"
  stats::lm(stats::as.formula(paste("y ~", rhs)), data = dat)
})
bic <- vapply(models, stats::BIC, numeric(1))
weight <- exp(-0.5 * (bic - min(bic)))
weight <- weight / sum(weight)

tibble::tibble(
  model = vapply(
    subsets,
    function(s) if (length(s) > 0) paste(s, collapse = " + ") else "(none)",
    character(1)
  ),
  weight = round(weight, 3)
)
#> # A tibble: 8 x 2
#>   model        weight
#>   <chr>         <dbl>
#> 1 (none)        0
#> 2 x1            0
#> 3 x2            0
#> 4 x3            0
#> 5 x1 + x2       0.848
#> 6 x1 + x3       0
#> 7 x2 + x3       0
#> 8 x1 + x2 + x3  0.152
```

Subtracting the smallest BIC before exponentiating leaves the normalized weights unchanged and avoids numerical underflow. The posterior inclusion probability of each predictor sums the weights of the models that contain it:

```
includes <- function(v) vapply(subsets, function(s) v %in% s, logical(1))
vapply(predictors, function(v) sum(weight[includes(v)]), numeric(1)) |>
  round(3)
#>    x1    x2    x3
#> 1.000 1.000 0.152
```

For the model-averaged slope of x_1, we approximate its posterior mean within each model by the least-squares estimate, taken as 0 in models that exclude x_1, and average with the posterior model probabilities as weights:

```
beta_x1 <- vapply(models, function(m) {
  cf <- stats::coef(m)
  if ("x1" %in% names(cf)) cf[["x1"]] else 0
}, numeric(1))
c(
  bma_slope_x1 = sum(weight * beta_x1),
  least_squares_x1_x2 = stats::coef(stats::lm(y ~ x1 + x2, data = dat))[["x1"]]
) |>
  round(3)
#>        bma_slope_x1 least_squares_x1_x2
#>               0.947               0.948
```

The predictors that affect the outcome, x_1 and x_2, have inclusion probabilities near 1, and the irrelevant x_3 a much smaller one. The models without x_1 get almost no weight, so the model-averaged slope of x_1 is close to its least-squares estimate in the model with x_1 and x_2 only. Both differ from the true slope 0.8 by sampling variation in this data set, not because of the averaging.

References

[1] A. J. Dobson and A. G. Barnett, *An introduction to generalized linear models*, 4th ed. CRC press, 2018. doi: 10.1201/9781315182780.

[2] B. Carpenter *et al.*, "Stan: A probabilistic programming language," *Journal of Statistical Software*, vol. 76, no. 1, pp. 1–32, 2017, doi: 10.18637/jss.v076.i01.

[3] M. Plummer, "JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling," in *Proceedings of the 3rd international workshop on distributed statistical computing*, 2003. Available: https://www.r-project.org/conferences/DSC-2003/Proceedings/Plummer.pdf

[4] M. Plummer, *JAGS version 4.3.0 user manual*. 2017. Available: https://sourceforge.net/projects/mcmc-jags/files/Manuals/4.x/

[5] M. D. Hoffman and A. Gelman, "The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo," *Journal of Machine Learning Research*, vol. 15, no. 47, pp. 1593–1623, 2014, Available: https://jmlr.org/papers/v15/hoffman14a.html

[6] Stan Development Team, *Stan user's guide*. 2026. Available: https://mc-stan.org/docs/stan-users-guide/

[7] Wikipedia contributors, "Bayesian hierarchical modeling — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Bayesian_hierarchical_modeling

[8] G. Schwarz, "Estimating the dimension of a model," *The Annals of Statistics*, vol. 6, no. 2, pp. 461–464, 1978, doi: 10.1214/aos/1176344136.

[9] Wikipedia contributors, "Bayesian information criterion — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Bayesian_information_criterion