Example Bayesian Analyses

Contents

Configuring R

Functions from these packages will be used throughout this document:

```
library(conflicted) # check for conflicting function definitions
# library(printr) # inserts help-file output into markdown output
library(rmarkdown) # Convert R Markdown documents into a variety of formats.
library(pander) # format tables for markdown
library(ggplot2) # graphics
library(ggfortify) # help with graphics
library(dplyr) # manipulate data
library(tibble) # `tibble`s extend `data.frame`s
library(magrittr) # `%>%` and other additional piping tools
library(haven) # import Stata files
library(knitr) # format R output for markdown
library(tidyr) # Tools to help to create tidy data
library(plotly) # interactive graphics
library(dobson) # datasets from Dobson and Barnett 2018
library(parameters) # format model output tables for markdown
library(haven) # import Stata files
library(latex2exp) # use LaTeX in R code (for figures and tables)
library(fs) # filesystem path manipulations
library(survival) # survival analysis
library(survminer) # survival analysis graphics
library(KMsurv) # datasets from Klein and Moeschberger
library(parameters) # format model output tables for
library(webshot2) # convert interactive content to static for pdf
library(forcats) # functions for categorical variables ("factors")
library(stringr) # functions for dealing with strings
```

```
library(lubridate) # functions for dealing with dates and times
library(broom) # Summarizes key information about statistical objects in tidy tibbles
library(broom.helpers) # Provides suite of functions to work with regression model 'broom::tidy()' t
```

Here are some R settings I use in this document:

```
rm(list = ls()) # delete any data that's already loaded into R

conflicts_prefer(dplyr::filter)
ggplot2::theme_set(
  ggplot2::theme_bw() +
        # ggplot2::labs(col = "") +
    ggplot2::theme(
      legend.position = "bottom",
      text = ggplot2::element_text(size = 12, family = "serif")))

knitr::opts_chunk$set(message = FALSE)
options('digits' = 6)

panderOptions("big.mark", ",")
pander::panderOptions("table.emphasize.rownames", FALSE)
pander::panderOptions("table.split.table", Inf)
conflicts_prefer(dplyr::filter) # use the `filter()` function from dplyr() by default
legend_text_size = 9
run_graphs = TRUE
```

This appendix collects worked Bayesian analyses, applying the concepts of the Bayesian-analysis appendix[1] and the simulation machinery of the MCMC appendix[2] to regression models we have already met in the frequentist setting. The structure parallels (Dobson and Barnett 2018, chap. 14). Each analysis specifies a model, places priors on its parameters, draws posterior samples by MCMC, and summarizes the posterior.

1 A first example: a single proportion

We begin with the simplest possible model — a single Bernoulli probability — fit with **JAGS** ("Just Another Gibbs Sampler"), which we drive from R. This example introduces the mechanics (specifying a model, supplying data, monitoring parameters, and inspecting draws) that the later analyses reuse.

This example demonstrates Bayesian inference using JAGS (Just Another Gibbs Sampler) for a simple Bernoulli model from Dobson's text (Dobson and Barnett 2018, chap. 13).

1.1 Load Required Packages

```
library(rjags)
library(runjags)
runjags::findJAGS()
#> [1] "/usr/bin/jags"
```

1.2 Run the JAGS Model

We'll use a simple Bernoulli model to estimate a probability parameter using Bayesian inference.

```
# JAGS chain initialization function
initsfunction <- function(chain) {
  stopifnot(chain %in% (1:4)) # max 4 chains allowed
```

[1] intro-bayes.qmd#sec-bayes-paradigms
[2] mcmc-methods.qmd#sec-mcmc-methods

```
  rng_seed <- (1:4)[chain]
  rng_name <- c(
    "base::Wichmann-Hill", "base::Marsaglia-Multicarry",
    "base::Super-Duper", "base::Mersenne-Twister"
  )[chain]
  return(list(".RNG.seed" = rng_seed, ".RNG.name" = rng_name))
}

# Generate sample data
set.seed(1)
data1 <- rbinom(n = 91, size = 1, prob = .6)

# Run JAGS model
jags_post0 <- run.jags(
  n.chains = 2,
  inits = initsfunction,
  model = system.file("extdata/model.dobson.jags", package = "rme"),
  data = list(r = data1, N = length(data1)),
  monitor = "p"
)
#> Compiling rjags model...
#> Calling the simulation using the rjags method...
#> Note: the model did not require adaptation
#> Burning in the model for 4000 iterations...
#> Running the model for 10000 iterations...
#> Simulation complete
#> Calculating summary statistics...
#> Calculating the Gelman-Rubin statistic for 1 variables....
#> Finished running the simulation
```

1.3 Examine Results

```
jags_post0$mcmc |> as.array() |> head()
#>       chain
#> iter       [,1]     [,2]
#>   5001 0.584687 0.550332
#>   5002 0.576621 0.562455
#>   5003 0.651983 0.628144
#>   5004 0.598258 0.604274
#>   5005 0.611863 0.583400
#>   5006 0.565635 0.606325
```

2 Binary outcomes: logistic regression

For a binary outcome $Y_i \sim \text{Bernoulli}(\pi_i)$ with $\text{logit}(\pi_i) = \tilde{x}_i^\top \tilde{\beta}$, the Bayesian analysis places a prior on the coefficient vector $\tilde{\beta}$ — typically independent, weakly informative normals such as $\beta_j \sim \text{N}(0, \sigma_0^2)$ with σ_0 large on the log-odds scale — and samples the posterior $p(\tilde{\beta} \mid \tilde{y})$ by MCMC (Dobson and Barnett 2018, chap. 14, p. 318). The posterior draws for each β_j are summarized by their mean and a 95% credible interval; exponentiating gives posterior summaries for the odds ratios. Unlike the frequentist fit, no large-sample normal approximation is needed: the credible interval is read directly from the posterior quantiles, and odds ratios or other nonlinear functions of $\tilde{\beta}$ are obtained simply by transforming the draws.

Exm

Example 2.1 (Bayesian logistic regression)**.** We simulate $N = 200$ observations from a logistic model with a single continuous predictor, intercept $\beta_0 = -0.5$ and slope $\beta_1 = 1.2$, then recover the coefficients by MCMC.

```
set.seed(2024)
N <- 200L
x <- rnorm(N)
beta0_true <- -0.5
beta1_true <- 1.2
y <- rbinom(N, size = 1, prob = plogis(beta0_true + beta1_true * 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)   # weakly informative: precision 0.01 = sd 10
  beta1 ~ dnorm(0, 0.01)
  OR <- exp(beta1)         # odds ratio per unit increase in x
}"

jags_logistic <- run.jags(
  model = logistic_model,
  data = list(y = y, x = x, N = N),
  monitor = c("beta0", "beta1", "OR"),
  n.chains = 2, inits = initsfunction,
  burnin = 1000, sample = 4000, method = "rjags"
)
#> Compiling rjags model...
#> Calling the simulation using the rjags method...
#> Adapting the model for 1000 iterations...
#> Burning in the model for 1000 iterations...
#> Running the model for 4000 iterations...
#> Simulation complete
#> Calculating summary statistics...
#> Calculating the Gelman-Rubin statistic for 3 variables....
#> Note: Unable to calculate the multivariate psrf
#> Finished running the simulation
summary(jags_logistic)[, c("Mean", "Lower95", "Upper95", "psrf")] |> round(3)
#>         Mean Lower95 Upper95 psrf
#> beta0 -0.657  -0.983  -0.302    1
#> beta1  1.205   0.819   1.622    1
#> OR     3.412   2.174   4.899    1
```

The posterior means recover $\beta_0 \approx -0.5$ and $\beta_1 \approx 1.2$, each 95% credible interval covers its true value, and the convergence statistic `psrf` ($\hat{R}$) is near 1. The odds-ratio summary comes for free by monitoring $\exp\{(\}\beta_1)$.

3 Survival analysis

Parametric survival models (for example, exponential or Weibull event times) admit a Bayesian treatment in which priors are placed on the baseline-hazard parameters and the regression coefficients, and the posterior is sampled by MCMC (Dobson and Barnett 2018, chap. 14, p. 330). Censoring is handled inside the model through the likelihood: an event contributes its density $\lambda(y)\,\mathrm{S}(y)$ and a censored observation contributes its survival probability $\mathrm{S}(y)$, exactly as in the frequentist

right-censored likelihood[3].

Exm

Example 3.1 (Bayesian exponential survival regression)**.** We simulate right-censored exponential survival times with a binary covariate (say, treatment) whose true log hazard ratio is $\beta_1 = -0.7$, and fit the model with the **zeros trick**, which lets us write the exact censored log-likelihood directly in JAGS.

```
set.seed(2026)
N <- 200L
x <- rbinom(N, size = 1, prob = 0.5)
beta0_true <- log(0.05)   # log baseline hazard
beta1_true <- -0.7        # log hazard ratio for x = 1
event_time <- rexp(N, rate = exp(beta0_true + beta1_true * x))
cens_time <- rexp(N, rate = 0.03)
y <- pmin(event_time, cens_time)
event <- as.integer(event_time <= cens_time)   # 1 = event, 0 = censored

surv_model <- "model {
  C <- 10000
  for (i in 1:N) {
    log(lambda[i]) <- beta0 + beta1 * x[i]
    # exponential log-likelihood, right-censored:
    loglik[i] <- event[i] * log(lambda[i]) - lambda[i] * y[i]
    phi[i] <- C - loglik[i]
    zeros[i] ~ dpois(phi[i])
  }
  beta0 ~ dnorm(0, 0.01)
  beta1 ~ dnorm(0, 0.01)
  HR <- exp(beta1)   # hazard ratio for x = 1 vs x = 0
}"

jags_surv <- run.jags(
  model = surv_model,
  data = list(y = y, x = x, event = event, N = N, zeros = rep(0, N)),
  monitor = c("beta0", "beta1", "HR"),
  n.chains = 2, inits = initsfunction,
  burnin = 1000, sample = 4000, method = "rjags"
)
#> Compiling rjags model...
#> Calling the simulation using the rjags method...
#> Adapting the model for 1000 iterations...
#> Burning in the model for 1000 iterations...
#> Running the model for 4000 iterations...
#> Simulation complete
#> Calculating summary statistics...
#> Calculating the Gelman-Rubin statistic for 3 variables....
#> Finished running the simulation
summary(jags_surv)[, c("Mean", "Lower95", "Upper95", "psrf")] |> round(3)
#>         Mean Lower95 Upper95  psrf
#> beta0 -3.115  -3.386  -2.861 1.000
#> beta1 -0.532  -0.933  -0.139 1.002
#> HR     0.600   0.374   0.843 1.002
```

The posterior for β_1 concentrates near the true -0.7, and the monitored $\exp\{(\}\beta_1)$ gives the hazard ratio with a credible interval that accounts for the censoring without any large-sample

[3] intro-to-survival-analysis.qmd#likelihood-with-censoring

approximation.

4 Random effects

The random-effects (multilevel) *model structure* — group-level parameters $\theta_j \sim \mathrm{N}(\mu, \tau^2)$ drawn from a common distribution — is the same one fit by maximum likelihood in (Dobson and Barnett 2018, chap. 11). What changes here is the *inference method*: the Bayesian treatment (the hierarchical model of the Bayesian-analysis appendix[4]) places priors on the hyperparameters μ and τ and samples their joint posterior together with the group-level parameters. The posterior **shrinks** each group's estimate toward the overall mean by an amount the data determine through τ, and — unlike the maximum-likelihood fit — the posterior for τ propagates the uncertainty in the between-group spread into every group-level summary (Dobson and Barnett 2018, chap. 14, p. 333).

Exm

Example 4.1 (Bayesian random-intercept model)**.** 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 noise $\sigma = 2$, then recover the hyperparameters.

[4] intro-bayes.qmd#sec-bayes-hierarchies

```r
set.seed(2025)
J <- 8L
n_j <- 12L
mu_true <- 5
tau_true <- 1.5
sigma_true <- 2
theta_true <- rnorm(J, mean = mu_true, sd = tau_true)
group <- rep(seq_len(J), each = n_j)
y <- rnorm(J * n_j, mean = theta_true[group], sd = sigma_true)

re_model <- "model {
  for (i in 1:N) {
    y[i] ~ dnorm(theta[group[i]], inv_sigma2)
  }
  for (j in 1:J) {
    theta[j] ~ dnorm(mu, inv_tau2)   # random intercepts
  }
  mu ~ dnorm(0, 1.0E-4)
  inv_sigma2 ~ dgamma(0.01, 0.01)
  inv_tau2 ~ dgamma(0.01, 0.01)
  sigma <- 1 / sqrt(inv_sigma2)   # within-group SD
  tau <- 1 / sqrt(inv_tau2)       # between-group SD
}"

jags_re <- run.jags(
  model = re_model,
  data = list(y = y, group = group, N = length(y), J = J),
  monitor = c("mu", "tau", "sigma"),
  n.chains = 2, inits = initsfunction,
  burnin = 1000, sample = 4000, method = "rjags"
)
#> Compiling rjags model...
#> Calling the simulation using the rjags method...
#> Note: the model did not require adaptation
#> Burning in the model for 1000 iterations...
#> Running the model for 4000 iterations...
#> Simulation complete
#> Calculating summary statistics...
#> Calculating the Gelman-Rubin statistic for 3 variables....
#> Finished running the simulation
summary(jags_re)[, c("Mean", "Lower95", "Upper95", "psrf")] |> round(3)
#>        Mean Lower95 Upper95  psrf
#> mu    5.556   4.838   6.262 1.000
#> tau   0.664   0.067   1.416 1.012
#> sigma 2.166   1.856   2.503 1.000
```

The posterior recovers the overall mean $\mu \approx 5$, the between-group SD $\tau \approx 1.5$, and the within-group SD $\sigma \approx 2$. Crucially, the credible interval for τ reports genuine uncertainty about the between-group spread — uncertainty a frequentist point estimate of the variance component discards.

5 Bayesian model averaging

When several candidate models are plausible, **Bayesian model averaging (BMA)** forms predictions by averaging over models, weighting each by its posterior probability rather than committing to a single "best" model (Dobson and Barnett 2018, chap. 14, p. 338). This propagates *model*

uncertainty — not just parameter uncertainty — into the final inference. Unlike the single-model fits above, BMA requires fitting and comparing a *collection* of models and computing their posterior weights, which connects directly to the model-comparison criteria of the MCMC appendix[5] and to our predictor-selection chapter[6].

Exm

Example 5.1 (A BIC approximation to Bayesian model averaging)**.** Marginal likelihoods are hard to compute, but the Bayesian information criterion provides a convenient approximation to the posterior model probability (Schwarz 1978): for model m, $p(m \mid \tilde{y}) \propto \exp\{-\frac{1}{2}\,\mathrm{BIC}_m\}$. We simulate data in which only x_1 and x_2 affect the outcome (with x_3 irrelevant), enumerate every subset of the three predictors, and form BIC-based model weights.

```
set.seed(2027)
N <- 120L
x1 <- rnorm(N)
x2 <- rnorm(N)
x3 <- rnorm(N) # irrelevant
y <- 1 + 0.8 * x1 + 0.5 * x2 + rnorm(N)
dat <- data.frame(y, x1, x2, x3)

predictors <- c("x1", "x2", "x3")
subsets <- unlist(
  lapply(0:3, function(k) combn(predictors, k, simplify = FALSE)),
  recursive = FALSE
)
models <- lapply(subsets, function(vars) {
  rhs <- if (length(vars)) paste(vars, collapse = " + ") else "1"
  lm(as.formula(paste("y ~", rhs)), data = dat)
})

# BIC approximation to posterior model probabilities:
bic <- vapply(models, BIC, numeric(1))
weight <- exp(-0.5 * (bic - min(bic)))
weight <- weight / sum(weight)
```

The **posterior inclusion probability** of each predictor is the total weight of the models that contain it:

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

and the **model-averaged** slope for x_1 averages each model's estimate (taken as 0 when x_1 is absent from that model):

```
beta_x1 <- vapply(models, function(m) {
  cf <- coef(m)
  if ("x1" %in% names(cf)) cf[["x1"]] else 0
}, numeric(1))
round(c(bma_slope_x1 = sum(weight * beta_x1)), 3)
#> bma_slope_x1
#>        0.947
```

[5] mcmc-methods.qmd#sec-mcmc-dic
[6] predictor-selection.qmd

> The real predictors x_1 and x_2 receive inclusion probabilities near 1, the irrelevant x_3 a small one, and the model-averaged slope for x_1 sits near its true value 0.8 — with the averaging across models, rather than a single selected model, accounting for which predictors belong.

References

Dobson, Annette J, and Adrian G Barnett. 2018. *An Introduction to Generalized Linear Models*. 4th ed. CRC press. https://doi.org/10.1201/9781315182780.

Schwarz, Gideon. 1978. "Estimating the Dimension of a Model." *The Annals of Statistics* 6 (2): 461–64. https://doi.org/10.1214/aos/1176344136.