Lecture: Bayesian Fundamentals

Bayesian fundamentals

Parameters as random variables

In the Bayesian world the unobserved quantities are assigned distributional properties and, therefore, become random variables in the analysis.

These distributions come in two basic flavors. If the distribution of the unknown quantity is not conditioned on fixed data, it is called prior distribution because it describes knowledge prior to seeing data.

Alternatively, if the distribution is conditioned on data that we observe, it is clearly updated from the unconditioned state and, therefore, more informed. This distribution is called posterior distribution.

Gill, J., & Witko, C. (2013). Bayesian analytical methods: A methodological prescription for public administration. Journal of Public Administration Research and Theory, 23(2), 457–494.

Contrast to the frequentist paradigm

Thus, the data is *fixed* and the parameter is a *random variable* that can be characterized in terms of distributional properties.

Contrast this with the frequentist the approach:

- The unknown *true* population parameter is fixed.
- The data is random: Each sample is a stochastic draw from a population.
- Therefore, any estimator/statistic is a random variable, too.
- The sampling distribution characterizes the probability distribution of such estimators and statistics, usually in terms of a normal distribution.

- Since the true mean and variance of the sampling distribution are unknown, we substitute them with the analogous sample estimates.

Bayesian updating

Bayesian updating follows the following idea:

$$\text{prior beliefs} \rightarrow \text{data} \rightarrow \text{posterior beliefs}$$

or equivalently,

$$p(\theta) \rightarrow y \rightarrow p(\theta|y)$$

Jackman, S. (2009). Bayesian Analysis for the Social Sciences. Wiley.

Prior beliefs can be vague – if we know (hardly) anything about a parameter θ prior to analyzing the data y – or specific, indicating we hold strong prior beliefs about θ even before seeing the data.

It is important to note that we express our beliefs about everything – prior, data, and posterior – *distributionally*:

- The prior and posterior distributions give probability distribution for θ before/after updating our knowledge by analyzing the data.
- A *likelihood function*, $p(y|\theta)$, gives a generative model that stipulates how the data y are linked to a parameter θ.

Let's think about this last point: How does it play out in the case of a single coin flip?

Bayes' theorem

Bayes' theorem lies at the core of Bayesian inference. It tells us how to update our prior beliefs upon seeing the data to arrive at posterior beliefs.

Bayes' theorem states that

$$p(\text{hypothesis}|\text{evidence}) = \frac{p(\text{hypothesis}) \times p(\text{evidence}|\text{hypothesis})}{p(\text{evidence})}$$

Bayes theorem: Proportional version

Since the data (or "evidence") is considered fixed and not a random variable, Bayes' theorem simplifies to its proportional form:

$$\underbrace{p(\text{hypothesis}|\text{evidence})}_{\text{posterior belief}} \propto \underbrace{p(\text{hypothesis})}_{\text{prior belief}} \times \underbrace{p(\text{evidence}|\text{hypothesis})}_{\text{likelihood function}}$$

or

$$p(\theta|\mathbf{y}) \propto p(\theta) \times p(\mathbf{y}|\theta)$$

Likelihood function

- Specification of a pdf or pmf for the *outcome*, y: $p(\mathbf{y}|\theta)$.
- Also called the data generating process (or the generative model) for y.
- Note that the likelihood function multiplies densities across *all* observations; e.g., a normal likelihood function is given by:

$$p(\mathbf{y}|\mu, \sigma) = \prod_{i=1}^{N} \text{Normal}\left(y_i|\mu_i, \sigma\right)$$

- This is what we mean mathematically when we use the shorthand $\mathbf{y} \sim \text{Normal}(\mu, \sigma)$

Prior distribution

- A distributional characterization of our belief about an unknown quantity (i.e., a *parameter*) prior to seeing the data: $p(\theta)$
- This includes statements about *family*, *support*, and *density*.
  - *Family*: A pdf (continuous parameters) or pmf (discrete parameters) that can plausibly generate the parameter values.
  - *Support*: Some parameters have constrained support: Probability parameters must be inside $[0, 1]$; variance parameters must be ≥ 0.

  - *Density*: A distributional characterization which values of the parameter we think are more or less likely to observe.

- The prior distribution can be
  - flat (i.e., uniformly distributed over the supported range)
  - purposefully vague, and thus, rather uninformative
  - weakly informative
  - specific and substantively informed (e.g., by previous research or expert assessment)

Posterior distribution

- Updating our distributional belief about θ given the data, $\mathbf{y}$: $p(\theta|\mathbf{y})$
- Follows the proportional version of Bayes' theorem: $p(\theta|\mathbf{y}) \propto p(\theta) \times p(\mathbf{y}|\theta)$
- Yields a weighthed combination of likelihood and prior
- The prior pulls the posterior density toward the center of gravity of the prior distribution
- As the data grows large, the likelihood becomes more influential:
  - one factor for $p(\theta)$, N factors for $p(y_i|\theta_i)$
  - we will see this analytically and using simulations later on

Coin flip experiment

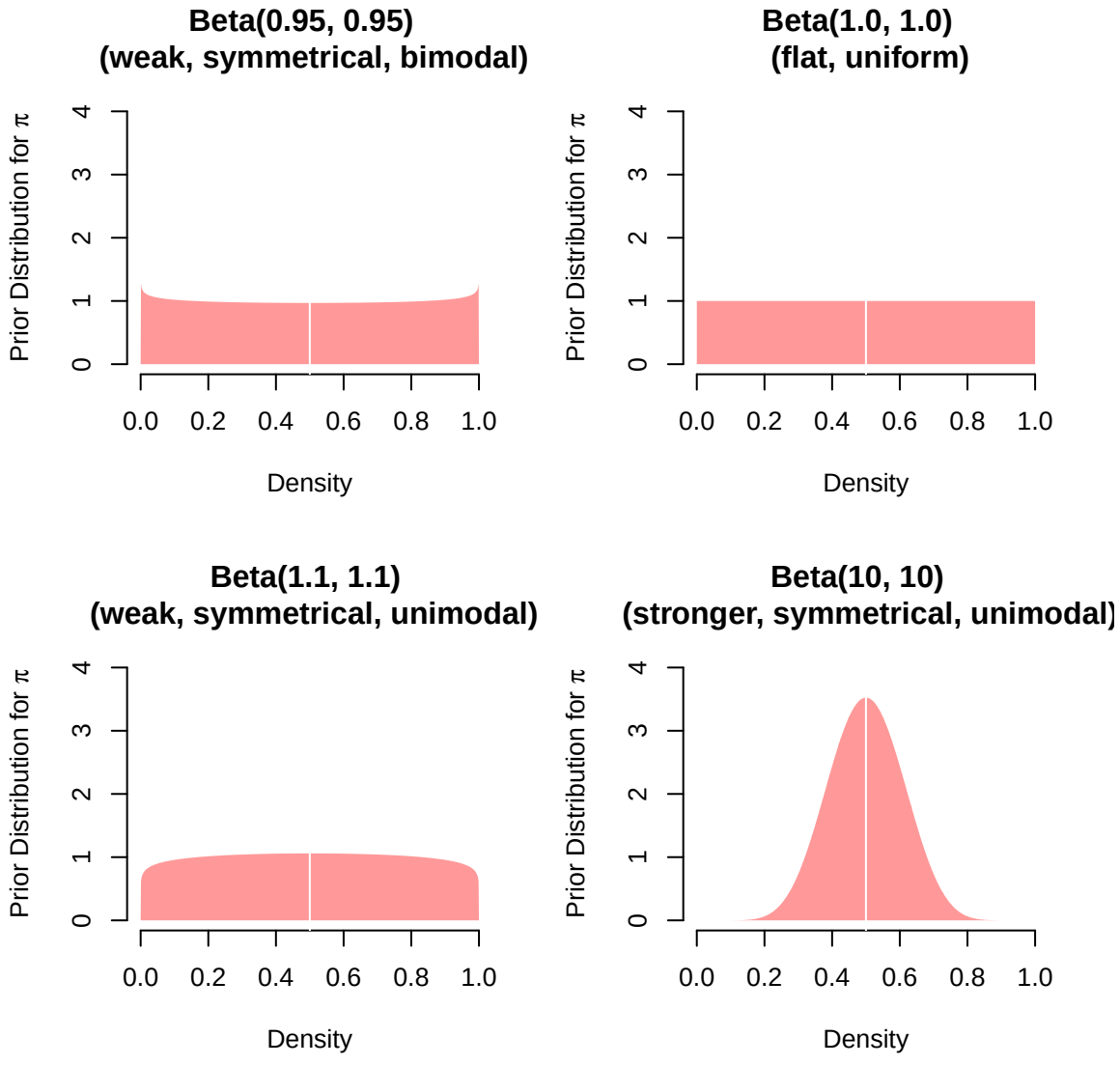
The experiment

Suppose we flip a coin up to N times:

- The fairness of a coin can be expressed through a *probability parameter*, π, that governs the probability that a coin flip produces heads (1) has opposed to tails (0)
- We start out with the belief that the coin is fair – that is, we consider it more probable that the coin is fair ($\pi \approx 0.5$) and less probable that it systematically over-produces either heads or tails
- Unbeknownst to us, the coin is far from fair – it is 4 times as likely to produce heads as it is to produce tails (that is, $\pi = 0.8$)
- We slowly learn about this in the process of flipping the coin and keeping score of the number of flips n and the number of heads k...

Analytical form: Prior distribution

- The *beta distribution* is a suitable candidate for characterizing our prior beliefs: $\pi \sim \text{beta}(a, b)$
- Characterized by two shape parameters, a and b
- a and b are *hyperparameters*: Known (or chosen) parameters that characterize a prior distribution.
- Constrained support: $\pi \in [0, 1]$
- pdf: $p(\pi) = \frac{\pi^{a-1}(1-\pi)^{b-1}}{\text{B}(a,b)}$



Analytical form: Likelihood

- Flipping one and the same coin n times is a series of Bernoulli trials
- The *binomial distribution* describes the corresponding data generating process: $k \sim \text{Binomial}(n, \pi)$
- pmf: $p(k|n, \pi) = \binom{n}{k}\pi^k(1-\pi)^{(n-k)}$

Analytical form: Posterior distribution

Remember:

$$p(\theta|\mathbf{y}) \propto p(\theta) \times p(\mathbf{y}|\theta)$$

So what does this mean in the present example?

$$p(\pi|n, k) \propto p(\pi) \times p(k|n, \pi)$$
$$p(\pi|n, k) \propto \frac{\pi^{a-1}(1-\pi)^{b-1}}{\text{B}(a, b)} \times \binom{n}{k}\pi^k(1-\pi)^{(n-k)}$$

Note that since we use the proportional version of Bayes' Law (i.e., we do not stipulate exact equality), we can drop any constant terms that do not involve our parameter of interest, π:

$$p(\pi|n, k) \propto \pi^{a-1}(1-\pi)^{b-1} \times \pi^k(1-\pi)^{(n-k)}$$

The rest, then, is easy: Following the rules of exponentiation, we add exponents for identical bases. This gives us our posterior distribution for π:

$$p(\pi|n, k) \propto \pi^{a+k-1}(1-\pi)^{b+n-k-1}$$

As you see, our posterior has the exact same form as our prior. It is a beta distribution with updated parameters

- $a' = a + k - 1$
- $b' = b + n - k - 1$

This property is called *conjugacy*: Prior and posterior are of the same family.

Now, take a moment to think about our analytical solution for the updated parameter:

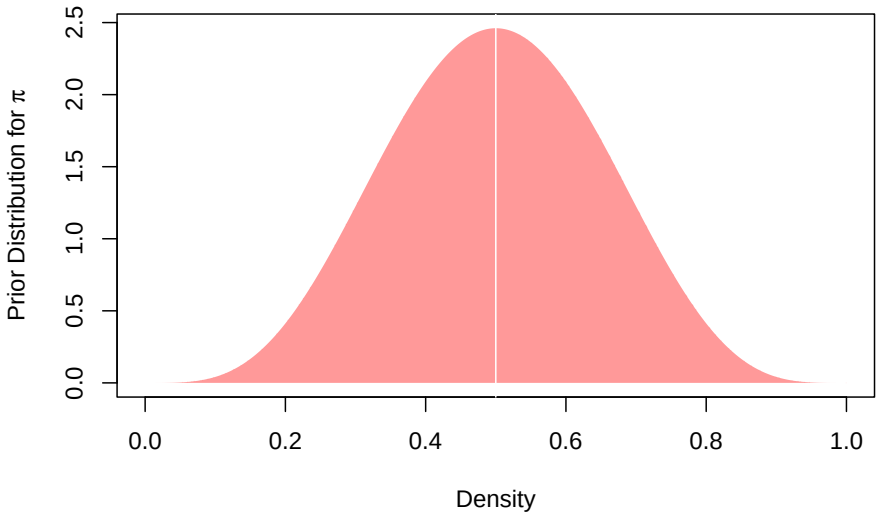
- What does it take for the data to dominate the prior?
- What if the prior is weak (e.g., $\pi \sim \text{beta}(a = 1, b = 1)$)?
- What if the prior is strong (e.g., $\pi \sim \text{beta}(a = 100, b = 100)$)?

Simulation

Prior distribution Code: Defining and plotting the prior distribution

```
len_pi <- 1001L                          ### number of candidate values for pi
pi <- seq(0, 1, length.out = len_pi) ### candidate values for pi
a <- b <- 5                              ### hyperparameters
prior <- dbeta(pi, a, b)                 ### prior distribution

## Plot
plot(                                    ### set up empty plot, specify labels
  pi, prior,
  type = 'n',
  xlab = "Density",
  ylab = expression(paste("Prior Distribution for ", pi))
)
polygon(                                 ### draw density distribution
  c(rep(0, length(pi)), pi),
  c(prior, rev(prior)),
  col = adjustcolor('red', alpha.f = .4),
  border = NA
)
abline(                                  ### add vertical at pi = 0.5
```

```
  v = .5,
  col = 'white'
)
```

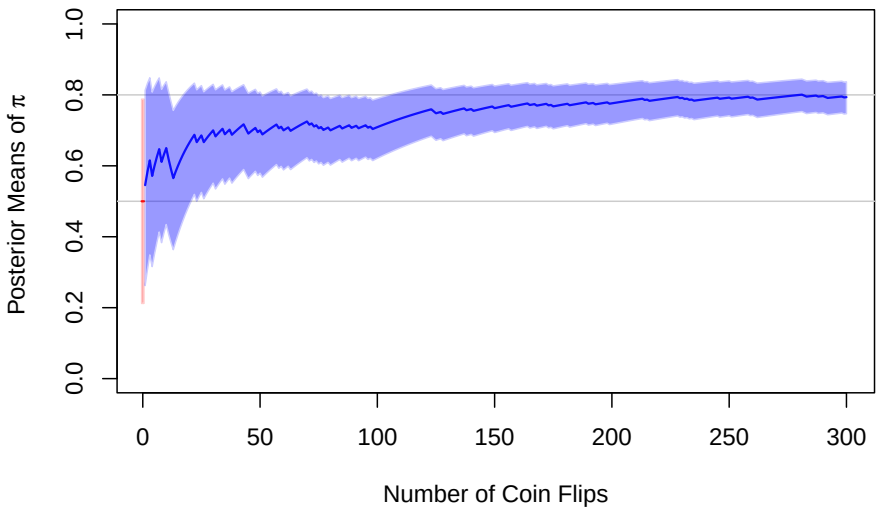


Posterior distribution Code: Simulating the experiment

```
set.seed(20210329)                    ### set seed for replicability
len_pi <- 1001L                       ### number of candidate values for pi
pi <- seq(0, 1, length.out = len_pi) ### candidate values for pi
a <- b <- 5                           ### hyperparameters
n <- 300                              ### num. of coin flips
pi_true <- .8                         ### true parameter
data <- rbinom(n, 1, pi_true)         ### n coin flips
posterior <- matrix(NA, 3L, n)        ### matrix container for posterior

for (i in seq_len(n)) {
  current_sequence <- data[1:i]       ### sequence up until ith draw
  k <- sum(current_sequence)          ### number of heads in current sequence
```

```r
  ##### Updating
  a_prime <- a + k
  b_prime <- b + i - k

  ### Analytical means and credible intervals
  posterior[1, i] <- a_prime / (a_prime + b_prime)
  posterior[2, i] <- qbeta(0.025, a_prime, b_prime)
  posterior[3, i] <- qbeta(0.975, a_prime, b_prime)
}

## Plot
plot(                                ### set up empty plot with labels
  1:n, 1:n,
  type = 'n',
  xlab = "Number of Coin Flips",
  ylab = expression(paste("Posterior Means of ",
                          pi,
                          sep = " ")),
  ylim = c(0, 1),
  xlim = c(1, n)
)
abline(                              ### reference line for the true pi
  h = c(.5, .8),
  col = "gray80"
)
rect(-.5, qbeta(0.025, 5, 5),        ### prior mean + interval at i = 0
     0.5, qbeta(0.975, 5, 5),
     col = adjustcolor('red', .4),
     border = adjustcolor('red', .2))
segments(-.5, .5,
         0.5, .5,
         col = adjustcolor('red', .9),
```

```
         lwd = 1.5)
polygon(                                ### posterior means + intervals
  c(seq_len(n), rev(seq_len(n))),
  c(posterior[2, ], rev(posterior[3, ])),
  col = adjustcolor('blue', .4),
  border = adjustcolor('blue', .2)
)
lines(
  seq_len(n),
  posterior[1, ],
  col = adjustcolor('blue', .9),
  lwd = 1.5
)
```



Note: After 300 coin flips, we have observed 241 heads, which is a proportion of 0.803. The posterior median is 0.794; the 95% credible interval is [0.747, 0.837].

MCMC algorithms

Analytical (classical) Bayesian inference

- As you may have noticed: Our coin flip example did *not* involve *any* numerical estimation algorithms.
- We simply observed the data, applied Bayes' Law, and analytically updated our parameters.
- This allowed us to retrieve a distributional characterization of our parameter of interest at each iteration of the coin flip series.
- The reasons why we could do this with ease is that this simple Binomial problem involved a single parameter π; i.e, we were dealing with a uni-dimensional *parameter space*.

The limits of analytical Bayesian inference

- Even in only slightly more intricate applications, Bayesian inference involves finding a *joint* posterior for *all* parameters in a model, i.e., finding a *multi-dimensional* parameter space.
- Inference on single parameters from a joint multi-dimensional parameter space requires that we retrieve the marginal posterior distribution from the joint posterior distribution.
- Marginalizing the joint multidimensional posterior distribution w.r.t. to a given a parameter gives the posterior distribution for that parameter. This requires *integrating* out all other parameters.
- For instance, when our joint posterior in a three-dimensional parameter space is $p(\alpha, \beta, \gamma)$, we need to obtain each marginal posterior akin to $p(\alpha) = \int_\beta \int_\gamma p(\alpha, \beta, \gamma) d\beta d\gamma$
- For complex multi-dimensional posterior distributions, finding analytical solutions through integration becomes cumbersome, if not outright impossible.

Numerical approximation via MCMC

That's where numerical approximation through Markov Chain Monte Carlo (MCMC) algorithms comes in:

- MCMC are iterative computational processes that explore and describe a posterior distribution.

- Developed in the 1980s and popularized in the 1990s, MCMC algorithms quickly eliminated the need for analytical marginalizations of single parameters from joint multi-dimensional posteriors.
- The core idea:
  - *Markov Chains* wander through, and take samples from, the parameter space.Following an initial warmup period, the Markov Chains will converge to high-density regions of the underlying posterior distribution (ergodicity).
  - The proportion of "steps" in a given region of multidimensional parameter space gives a stochastic simulation of the posterior probability density.
  - This yields a numerical approximation of the underlying posterior distribution, much like Monte Carlo simulations of MLE parameters yield numerical approximations of the underlying sampling distribution.

(Some) MCMC Algorithms

1. **Gibbs**: Draws iteratively and alternatively from the conditional conjugate distribution of each parameter.
2. **Metropolis-Hastings**: Considers a single multidimensional move on each iteration depending on the quality of the proposed candidate draw.
3. **Hamiltonian Monte Carlo (HMC)**, used in Stan:

The Hamiltonian Monte Carlo algorithm starts at a specified initial set of parameters θ; in Stan, this value is either user-specified or generated randomly. Then, for a given number of iterations, a new momentum vector is sampled and the current value of the parameter θ is updated using the leapfrog integrator with discretization time ϵ and number of steps L according to the Hamiltonian dynamics. Then a Metropolis acceptance step is applied, and a decision is made whether to update to the new state $(\theta^*, \rho*)$ or keep the existing state.

Source: Stan Reference Manual, Section 14.1

Resource: Animated visualizations of several MCMC algorithms

Chi Feng hosts animated visualizations of several MCMC algorithms on his website. Worth a visit!

Gibbs sampler

In a nutshell

- MCMC algorithms can be complex.
- Here, we will illustrate the intuition of sampling from the joint posterior of all parameters using the arguably most straightforward MCMC algorithm: The *Gibbs sampler*.
- Remember that Gibbs draws iteratively and alternatively from the conditional conjugate distribution of each parameter.
- We want to perform inference on a variable y, of which we have N observations.
- We stipulate that the data-generating process that produces y is normal: $\mathbf{y} \sim \mathrm{N}(\mu, \sigma^2)$. This yields a *two-dimensional parameter space* – i.e., a bivariate posterior distribution.

Application

We will focus on the variable `sup_afd` from the data set `gles`.

Let's pretend our prior belief is very uninformed:

- We don't know how (un)popular the AfD is in the German electorate
- But we know that individual support is measured on a -5 to 5 scale
- Our prior belief for μ should thus be agnostic as to whether people like or dislike the AfD and sufficiently vague to allow for the possibility that we may be wrong: $\mu \sim \text{Normal}(\theta = 0, \omega^2 = 4)$ (mean θ and variance ω^2 are hyperparameters for the normal prior distribution of μ)
- Our prior belief for τ will also be vague: $\sigma^2 \sim \text{Gamma}^{-1}(\alpha = 2, \beta = 10)$ (shape α and rate β are hyperparameters for the inverse Gamma prior distribution of τ)
- We have no prior belief about the dependence of both parameters and hence specify independent prior distributions

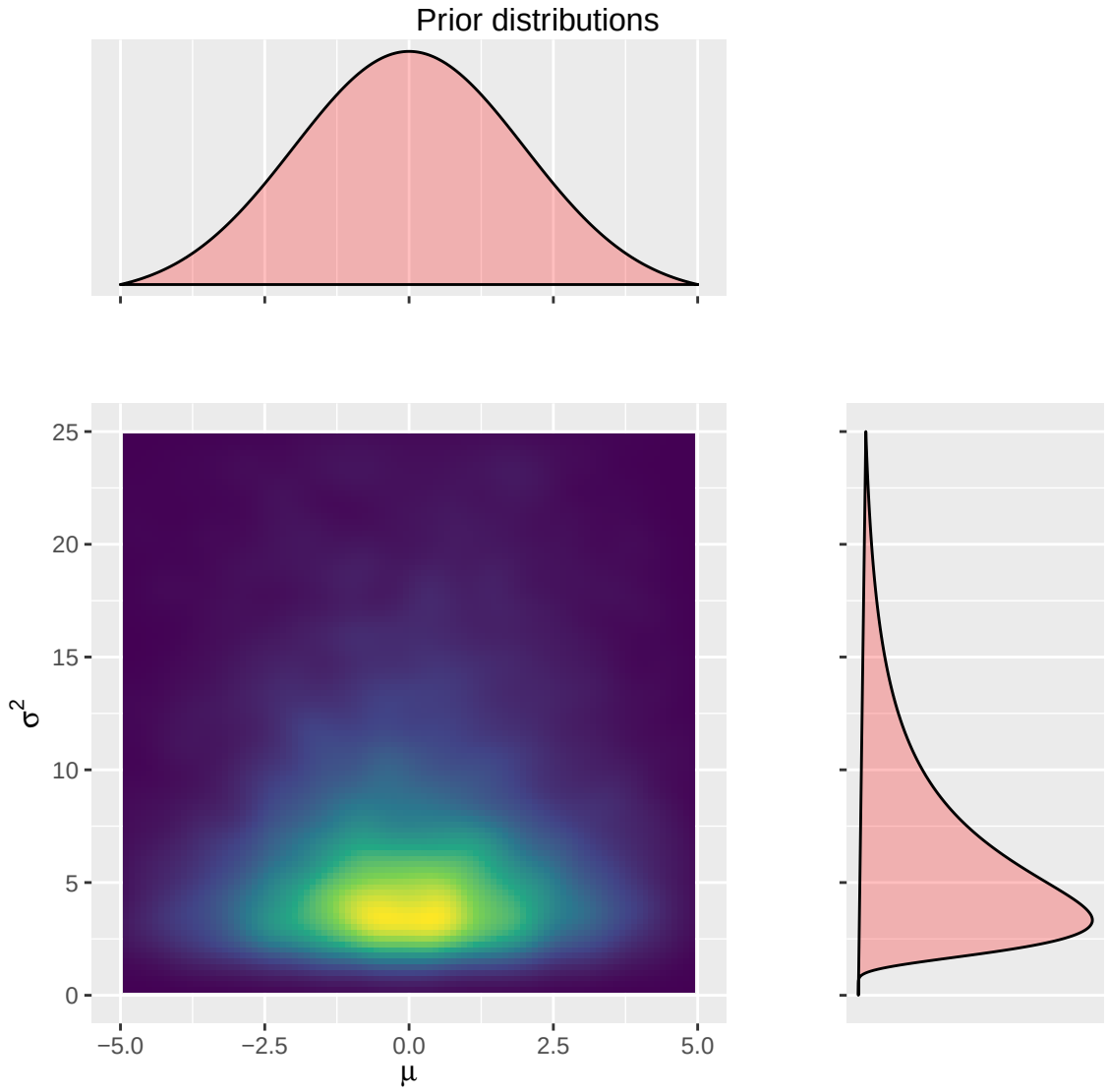
Analytical prerequisites

- A Gibbs sampler requires analytical solutions – mathematical formulas – for the *conditional* posterior distributions of the two parameters from whose marginal posteriors we would like to sample.
- Note that this does *not* involve marginalizing out the "unwanted" parameters; instead, we will derive the posteriors of μ and σ^2 as conditional functions of each other, respectively.
- Put simply, the updated posterior of μ depends on values of σ^2, and the updated posterior of σ^2 depends on values of μ.

Sampling

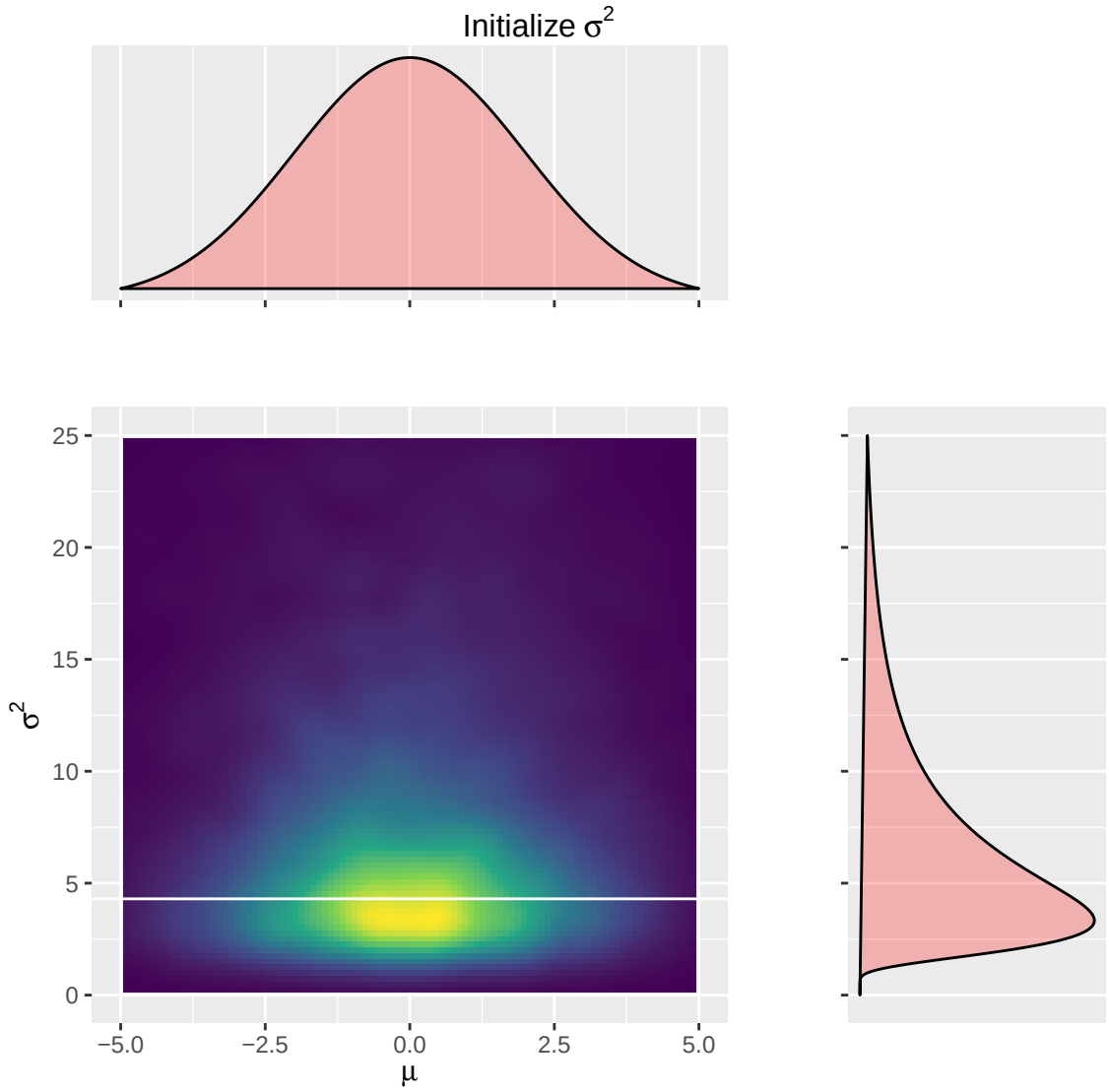
- We first initialize σ_1^2 by taking a random draw from its prior distribution.
- We then initialize μ_1 by taking a draw from its updated posterior distribution, conditional on σ_1^2
- We then sample iteratively and alternatively:
  - Draw σ_2^2 from its updated posterior distribution, conditional on μ_1
  - Draw μ_2 from its updated posterior distribution, conditional on σ_2^2
  - Draw σ_s^2 from its updated posterior distribution, conditional on μ_{s-1} for $s \in \{3, ..., S\}$
  - Draw μ_s from its updated posterior distribution, conditional on σ_s^2 for $s \in \{3, ..., S\}$
- Eventually, after a sufficiently large number of simulations S, the sampler converges to its *posterior target distribution.*
- Once converged, any draws sampled from the posterior target distributions give accurate *numerical simulations* of the joint posterior distribution of μ and σ^2.

Step 1: Understanding the prior distributions of μ and σ^2



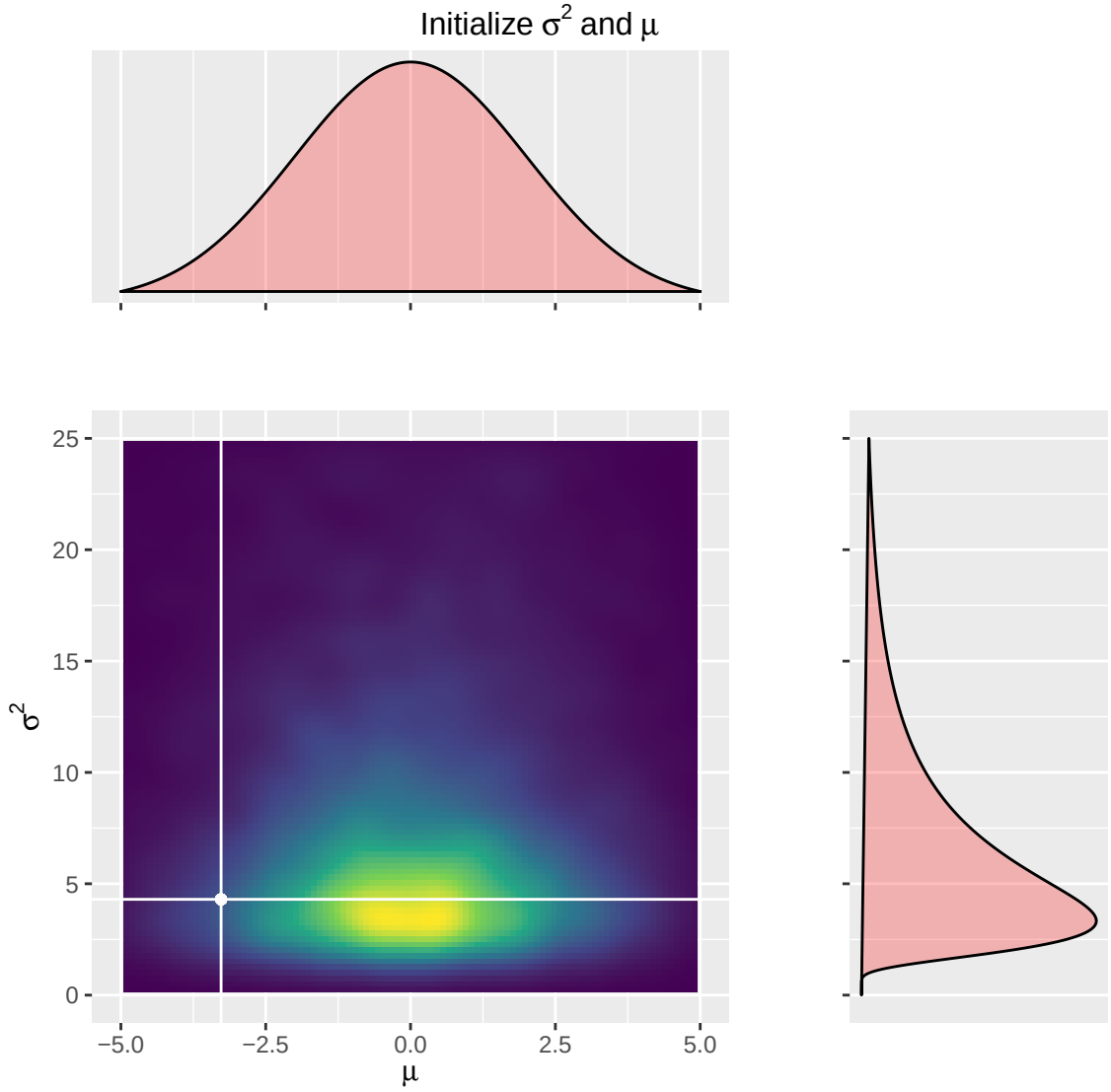
Step 2a: Initializing σ^2

We draw σ_1^2 from its prior distribution:



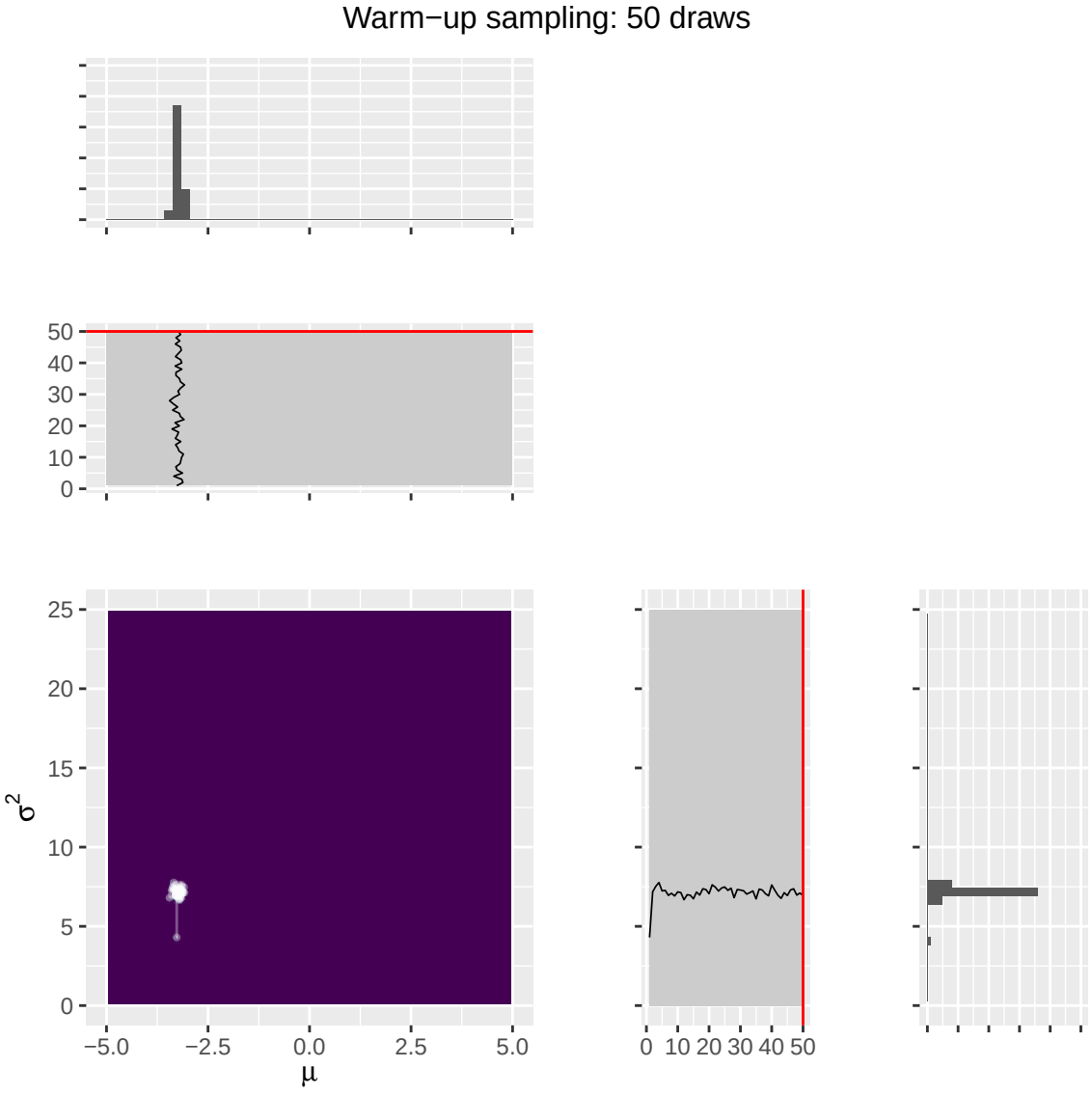
Step 2b: Initializing μ

We draw μ_1 from the conditional conjugate posterior, given σ_1^2:



Step 3: Sampling warm-up draws

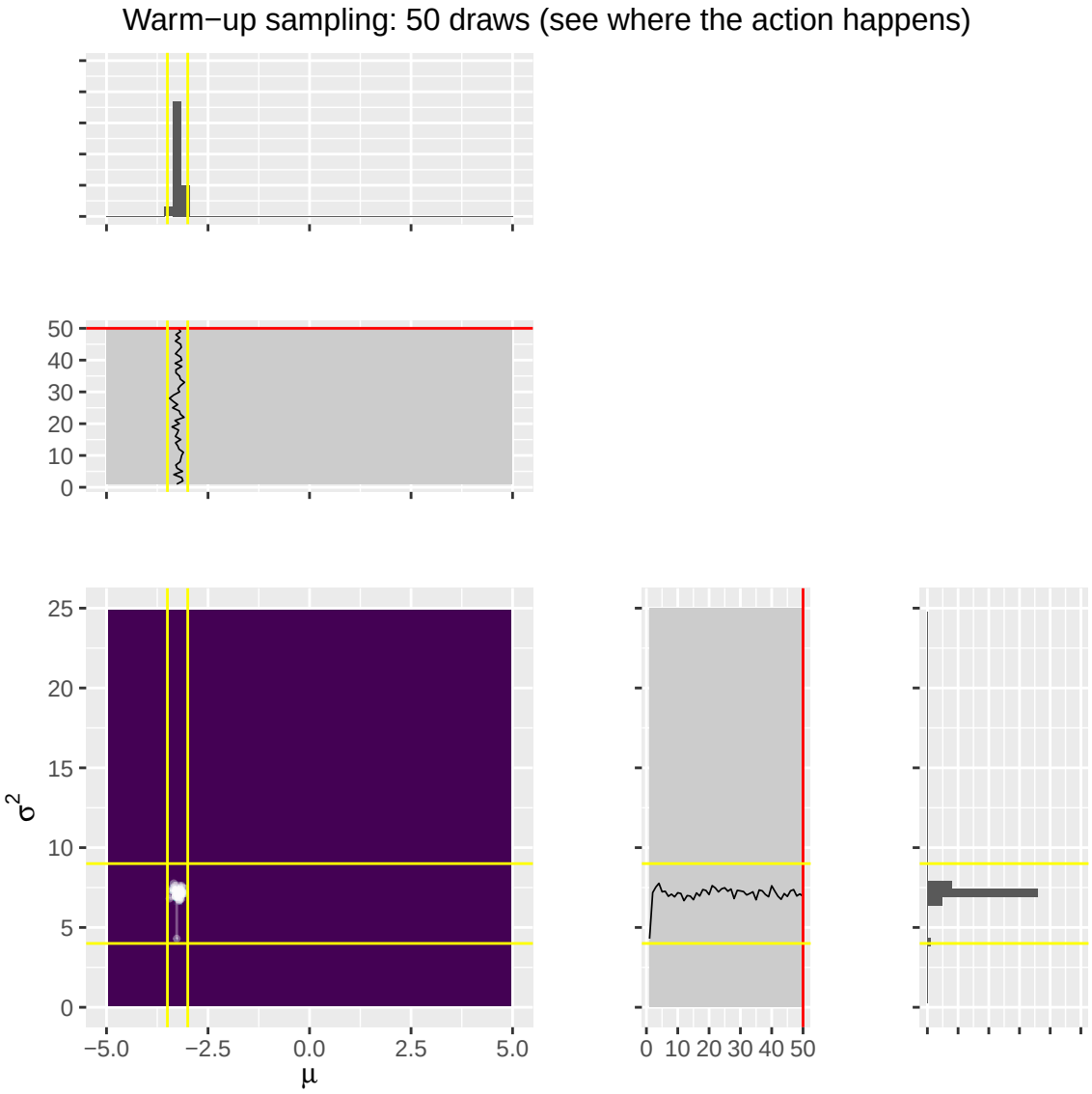
We run the Gibbs sampler for an initial period of fifty draws.



Posterior regions of interest

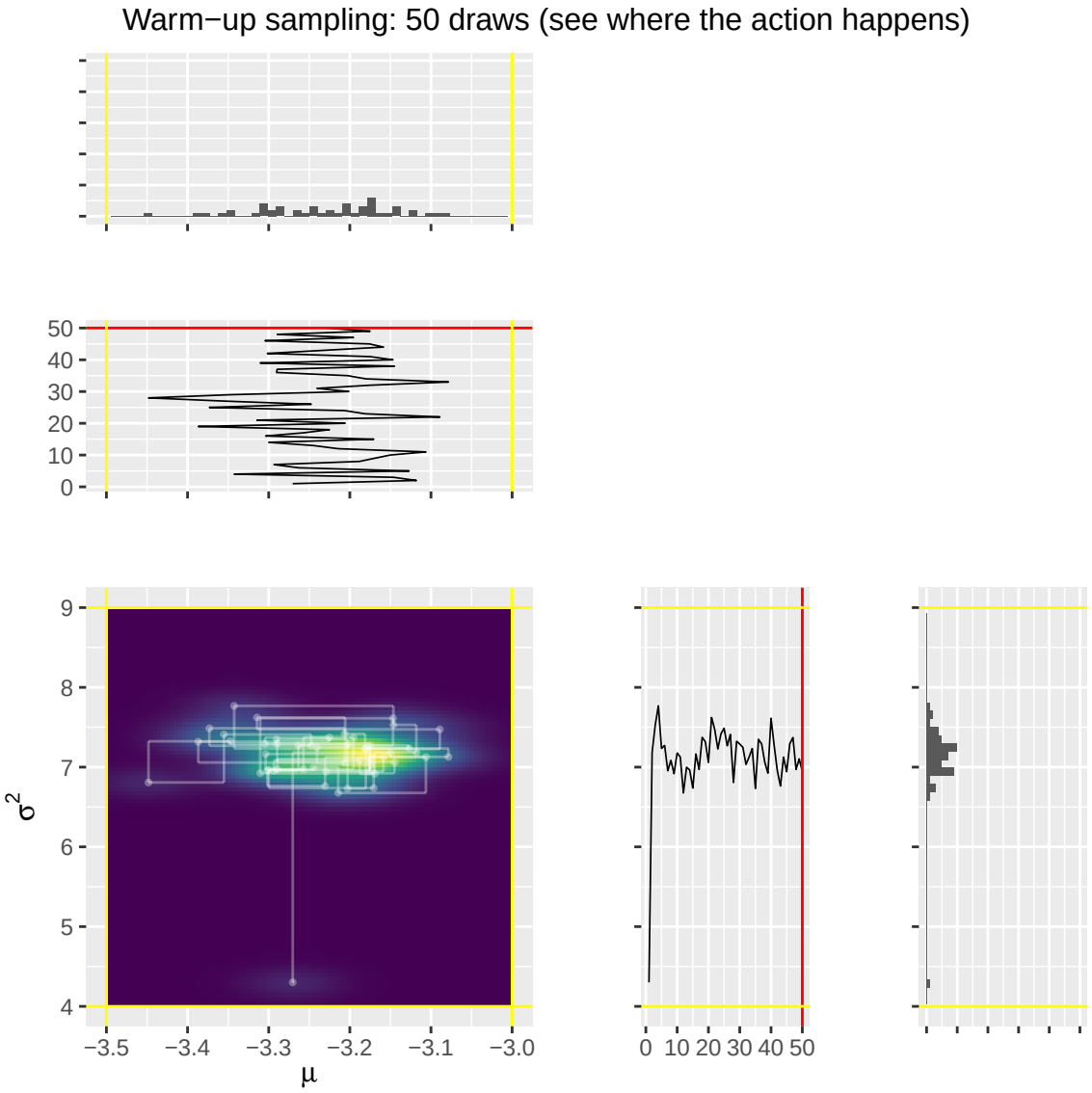
Upon closer inspection, we see that the Gibbs sampler immediately converges to the posterior target distribution.

Unlike our very vague prior, the posterior conveys specific knowledge: All draws are within the yellow lines.

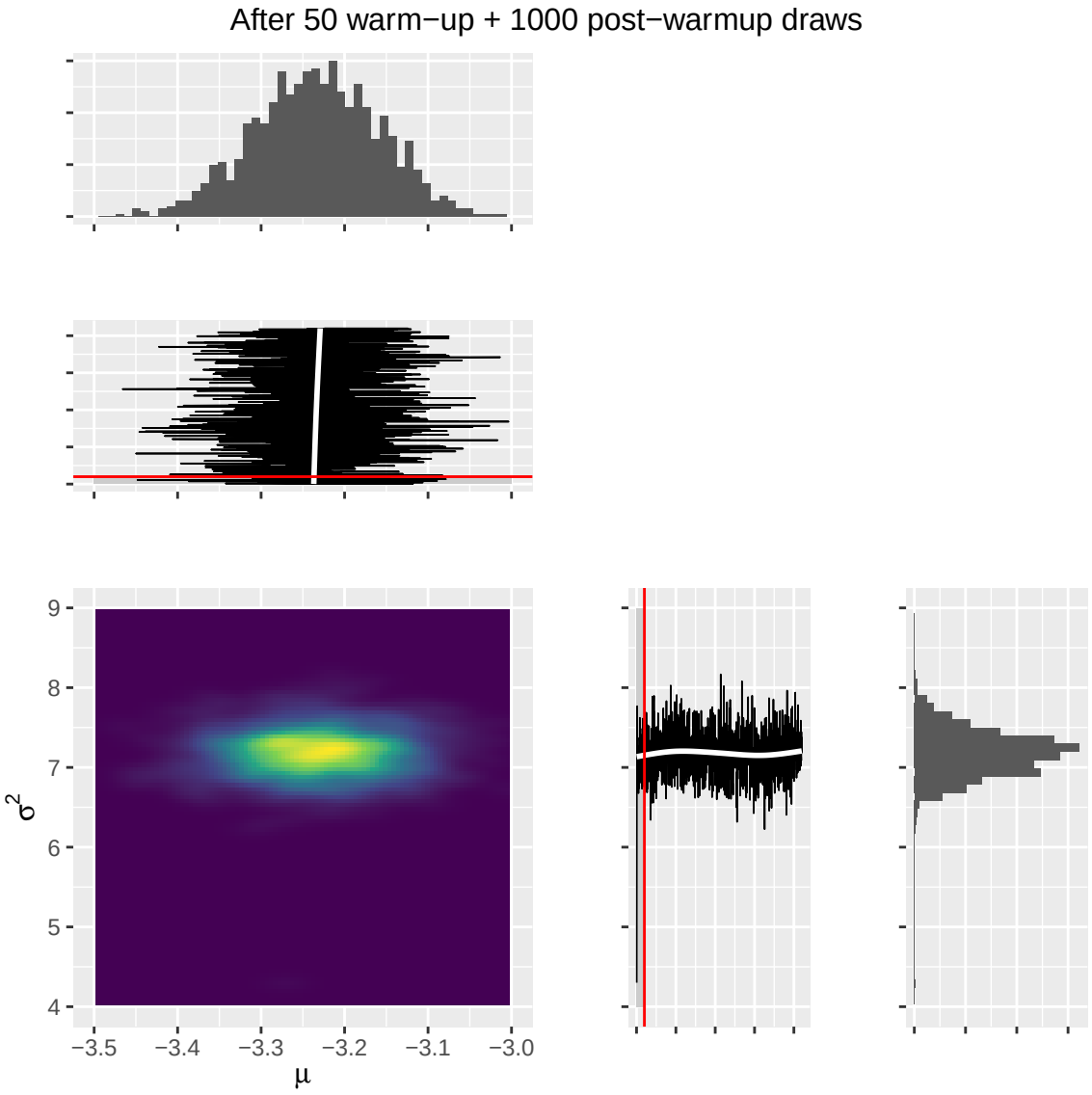


Zooming in

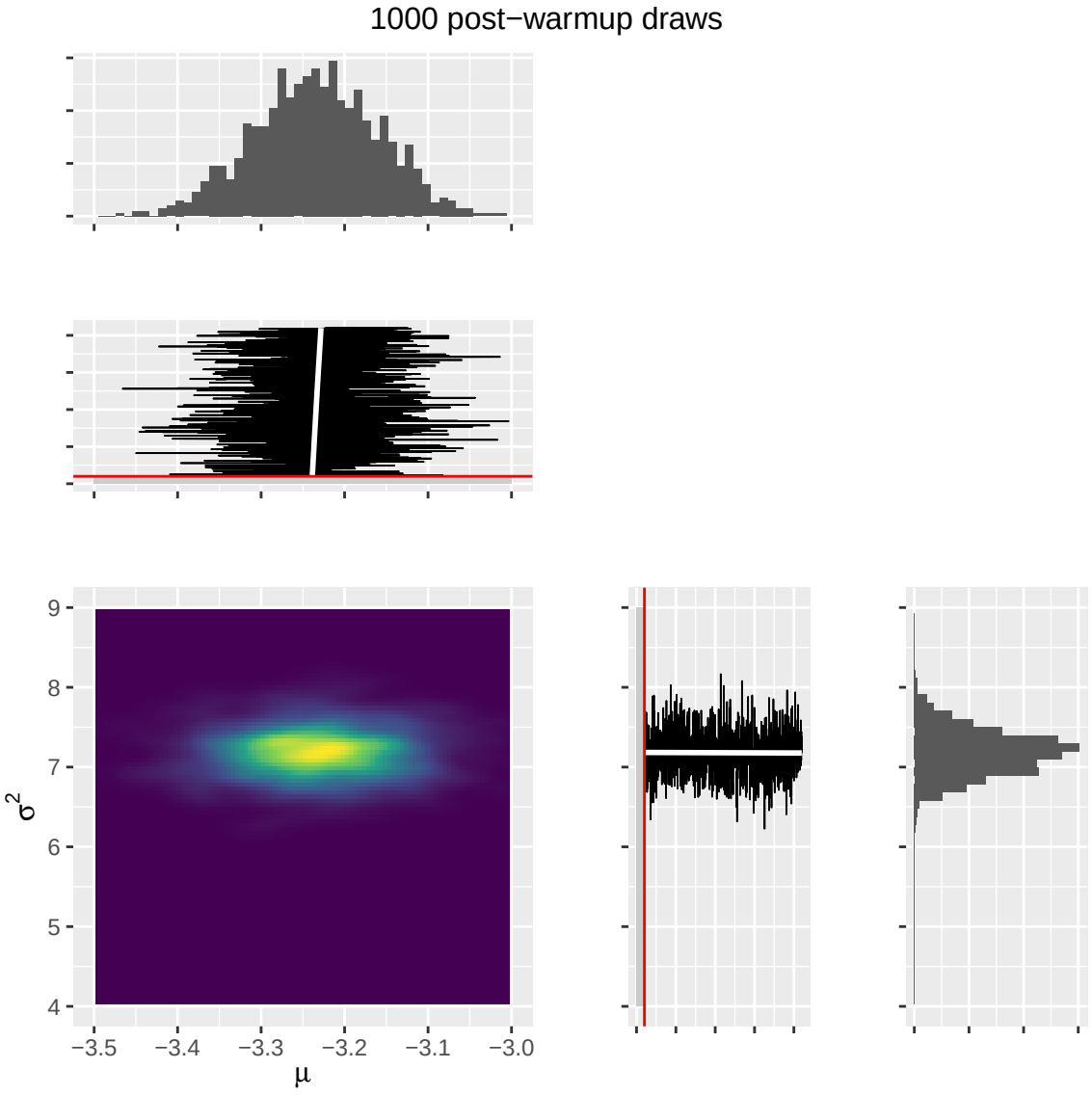
We therefore "zoom in" on the rectangular area inbetween the yellow lines.


Warm−up sampling: 50 draws (see where the action happens)
50
40
30
20
10
0
9
8
7
6
5
4
σ2
−3.5
−3.4
−3.3
−3.2
−3.1
−3.0
μ
0 10 20 30 40 50

Step 4: Run the sampler for at least 1000 post-warmup iterations



Step 5: Discard the warmup draws



Step 6: Summarize the marginal posterior distributions via distributional summaries

```
round(apply(posterior_data, 2, median), 2)
```

Posterior medians as Bayesian analogues to point estimates

```
##     mu sigma2
##  -3.23   7.19
```

```r
round(apply(posterior_data, 2, sd), 2)
```

Posterior standard deviations as Bayesian analogues to standard errors

```
##     mu sigma2
##   0.08   0.28
```

```r
round(apply(posterior_data, 2, quantile, c(.025, .975)), 2)
```

Quantile-based credible intervals as Bayesian uncertainty intervals

```
##          mu sigma2
## 2.5%  -3.38   6.63
## 97.5% -3.10   7.74
```

```r
mean(posterior_data$mu < 0)
```

Posterior proportions as Bayesian directional ("one-tailed") hypothesis tests

```
## [1] 1
```

```r
mean(posterior_data$mu > -3.33)
```

```
## [1] 0.901
```

Convergence diagnostics

Why diagnose?

MCMC algorithms use iterative algorithms to explore posterior distributions and to produce numerical approximations thereof.

However, even with appropriately specified models and algorithms, we can never know a priori if and when a chain has converged to its target distribution. We must thus rely on *convergence diagnostics*.

Important: Convergence diagnostics cannot show or prove convergence. They can only show signs of non-convergence!

How to diagnose

To conclude that the post-warmup draws of our sampler in fact explore the target distribution, we want to run *multiple, independent chains* on the same data and show two things:

1. Every chain is in a stationary state (i.e., does not "wander off" the target distribution) – no trending *within-chain variance*
2. All chains are in the same stationary state (i.e., no convergence to different target distributions given identical data) – no systematic *between-chain variance*

Generic diagnostics

Generic diagnostics (see Gill 2015, Ch. 14.3) include:

1. **Potential scale reduction statistic** $\hat{R}$ (a.k.a. Gelman-Rubin convergence diagnostic)

$$\widehat{Var}(\theta) = (1 - \frac{1}{\texttt{n}_{\texttt{iter}}}) \underbrace{\left(\frac{1}{\texttt{n}_{\texttt{chains}}(\texttt{n}_{\texttt{iter}} - 1)} \sum_{j=1}^{\texttt{n}_{\texttt{chains}}} \sum_{i=1}^{\texttt{n}_{\texttt{iter}}} (\theta_{ij} - \bar{\theta_j})^2 \right)}_{\text{Within chain var}} + \frac{1}{\texttt{n}_{\texttt{iter}}} \underbrace{\left(\frac{\texttt{n}_{\texttt{iter}}}{\texttt{n}_{\texttt{chains}} - 1} \sum_{j=1}^{\texttt{n}_{\texttt{chains}}} (\bar{\theta_j} - \bar{\bar{\theta}})^2 \right)}_{\text{Between chain var}}$$

   - low values indicate that chains are stationary (convergence to target distribution within chains)
   - low values indicate that chains mix (convergence to same target distribution across chains)

2. **Geweke Time-Series Diagnostic**: Compare non-overlapping post-warmup portions of each chain to test within-convergence
3. **Heidelberger and Welch Diagnostic**: Compare early post-warmup portion of each chain with late portion to test within-convergence

4. **Raftery and Lewis Integrated Diagnostic**: Evaluates the full chain of a pilot run (requires that `save_warmup = TRUE`) to estimate minimum required length of warmup and sampling

These are implemented as part of the `coda` package (Output Analysis and Diagnostics for MCMC).

Visual diagnostics

The most widespread visual diagnostics are:

1. **Traceplots**: Visually inspect if chains are stationary and have converged to the same distribution
2. **Autocorrelation plots**: Visually inspect if the chain is sluggish in exploring the parameter space.

Application

In the following, we will use multiple chain runs of our sampler in conjunction with the `coda` package to check for signs of non-convergence.

Note that `coda` functions require that we combine our chains into `mcmc.list` objects.

Raftery and Lewis Integrated Diagnostic

The Raftery-Lewis diagnostic takes a single chain, including warm-up draws, to estimate the minimum required length of warmup and sampling runs:

```
# Example: Gill 2015, p. 503
# If we want a 95% credible interval around the median
# with reliability between 92.5% and 97.5%, we need:
q <- 0.5    # quantile of interest
r <- 0.0125 # margin of error
s <- 0.95   # desired reliability
```

```r
## The recommend length for the pilot run:
n <- ceiling((qnorm(.5 * (s + 1)) * sqrt(q * (1 - q)) / r) ^ 2)

# Pilot run
draws_pilot <-
  draw_from_posterior(
    theta = 0,
    omega = .1,
    alpha = 20,
    beta = 200,
    n_warmup = 0,
    n_draws = n,
    data = gles$sup_afd,
    keep_warmup = TRUE
  )

# Save as mcmc
draws_pilot <- as.mcmc(simplify2array(draws_pilot))

# Diagnose
raftery.diag(
  draws_pilot,
  q = q,
  r = r,
  s = s
)
```

```
##
## Quantile (q) = 0.5
## Accuracy (r) = +/- 0.0125
## Probability (s) = 0.95
##
```

```
##       Burn-in  Total Lower bound  Dependence
##       (M)      (N)   (Nmin)       factor (I)
##  mu   2        6278  6147         1.020
##  tau  2        5906  6147         0.961
```

Gelman-Rubin, Geweke, and Heidelberger-Welch diagnostics

We will use the recommended run-length from the Raftery-Lewis diagnostic for four independent runs of our sampler.

We will ensure that our chains run independently (i.e., using different starting values and different random number sequences) by setting different seed:

```
seeds <- sample(10001:99999, 4)
draws_multiple_chains <- lapply(seeds,
                                function(seed) {
                                  as.mcmc(simplify2array(
                                    draw_from_posterior(
                                      theta = 0,
                                      omega = .25,
                                      alpha = 2,
                                      beta = 10,
                                      n_warmup = 200,
                                      n_draws = 6147,
                                      data = gles$sup_afd,
                                      keep_warmup = FALSE,
                                      seed = seed
                                    )
                                  ))
                                })

# Save as mcmc.list
draws_multiple_chains <- as.mcmc.list(draws_multiple_chains)
```

```r
# Diagnose
coda::gelman.diag(draws_multiple_chains, autoburnin = FALSE)
```

```
## Potential scale reduction factors:
##
##     Point est. Upper C.I.
## mu           1          1
## tau          1          1
##
## Multivariate psrf
##
## 1
```

```r
coda::geweke.diag(draws_multiple_chains, frac1 = .1, frac2 = .5)
```

```
## [[1]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##     mu    tau
## 0.9192 0.7007
##
##
## [[2]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##      mu     tau
## -0.1502  0.2664
##
```

```
##
## [[3]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##       mu      tau
## -0.05762 -0.96938
##
##
## [[4]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##    mu   tau
## 1.192 1.112
```

```
coda::heidel.diag(draws_multiple_chains, pvalue = .1)
```

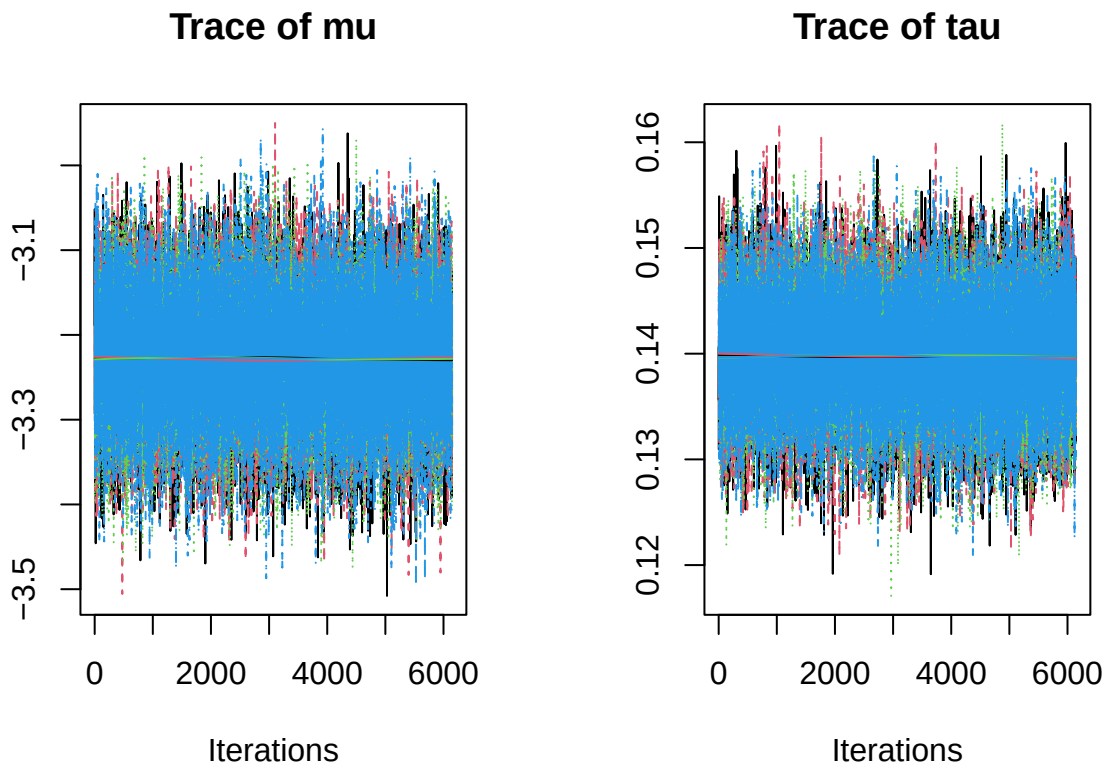
```
## [[1]]
##
##     Stationarity start     p-value
##     test         iteration
## mu  passed       1         0.204
## tau passed       1         0.695
##
##     Halfwidth Mean  Halfwidth
##     test
## mu  passed    -3.23 0.001810
## tau passed     0.14 0.000133
##
```

```
## [[2]]
##
##     Stationarity start     p-value
##     test         iteration
## mu  passed       1         0.771
## tau passed       1         0.332
##
##     Halfwidth Mean  Halfwidth
##     test
## mu  passed    -3.23 0.001843
## tau passed     0.14 0.000138
##
## [[3]]
##
##     Stationarity start     p-value
##     test         iteration
## mu  passed       1         0.321
## tau passed       1         0.639
##
##     Halfwidth Mean  Halfwidth
##     test
## mu  passed    -3.23 0.001843
## tau passed     0.14 0.000136
##
## [[4]]
##
##     Stationarity start     p-value
##     test         iteration
## mu  passed       1         0.653
## tau passed       1         0.341
##
##     Halfwidth Mean  Halfwidth
```

```
##      test
## mu   passed    -3.23 0.001851
## tau passed     0.14 0.000143
```

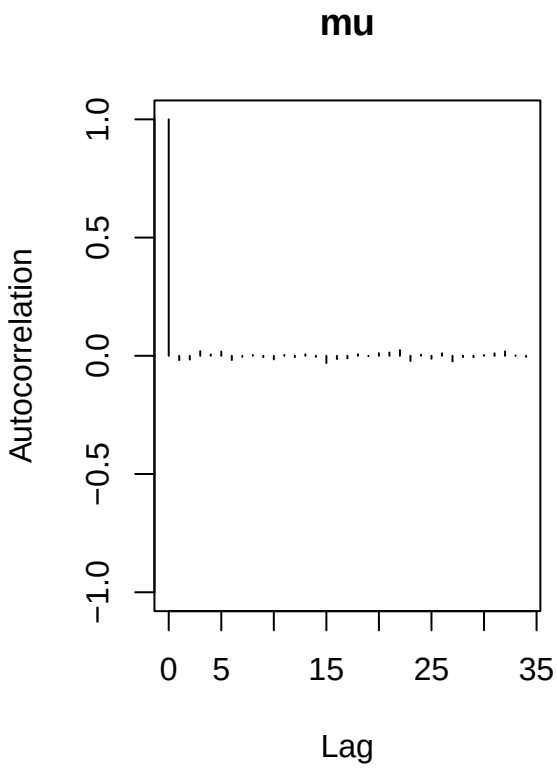
Trace plots

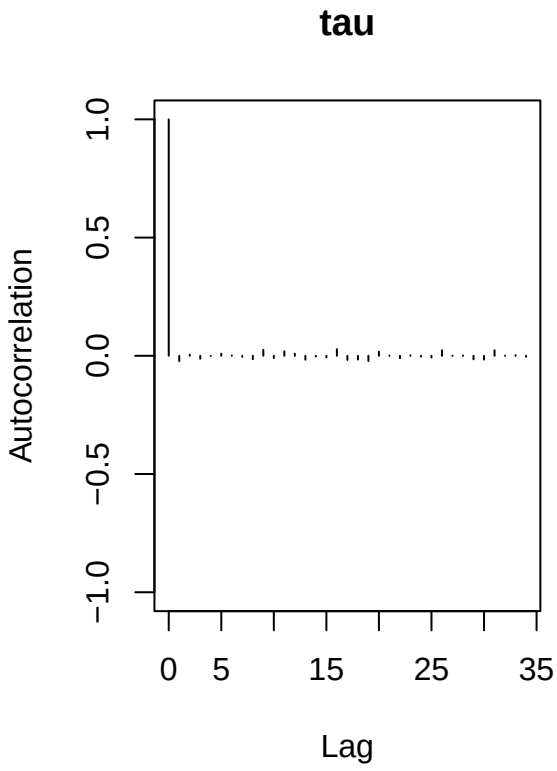
```
par(mfrow = c(1, 2))
coda::traceplot(draws_multiple_chains, smooth = TRUE)
```

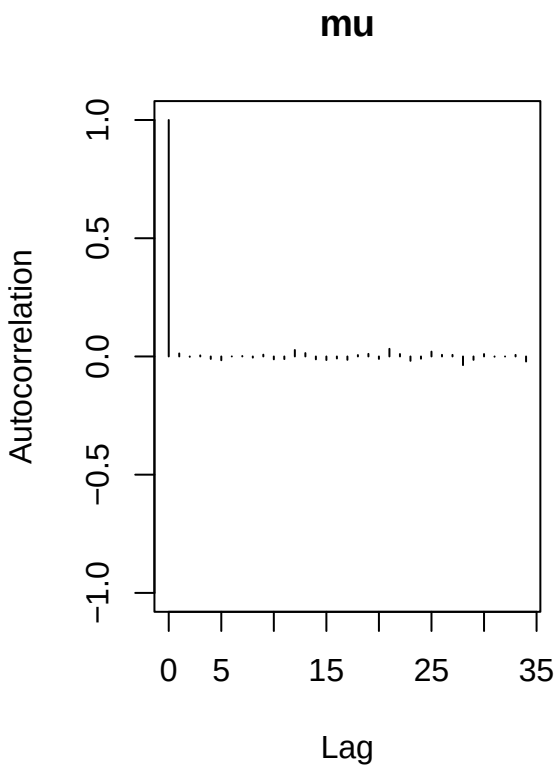


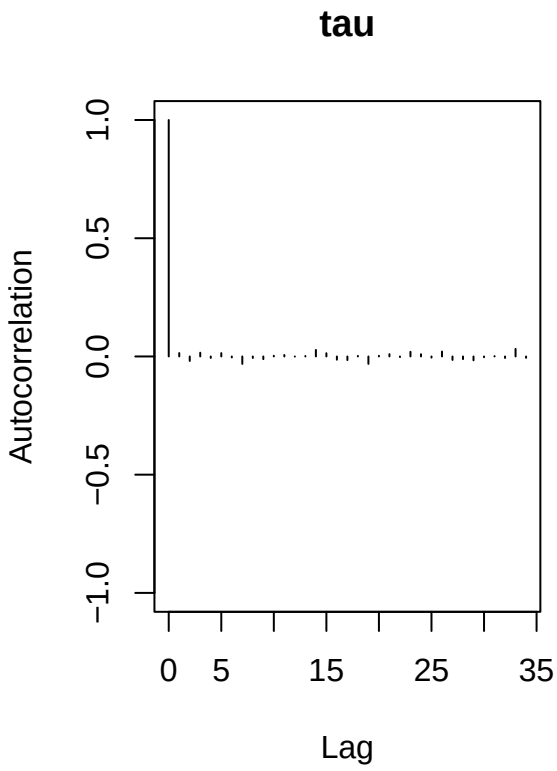
Autocorrelation plots

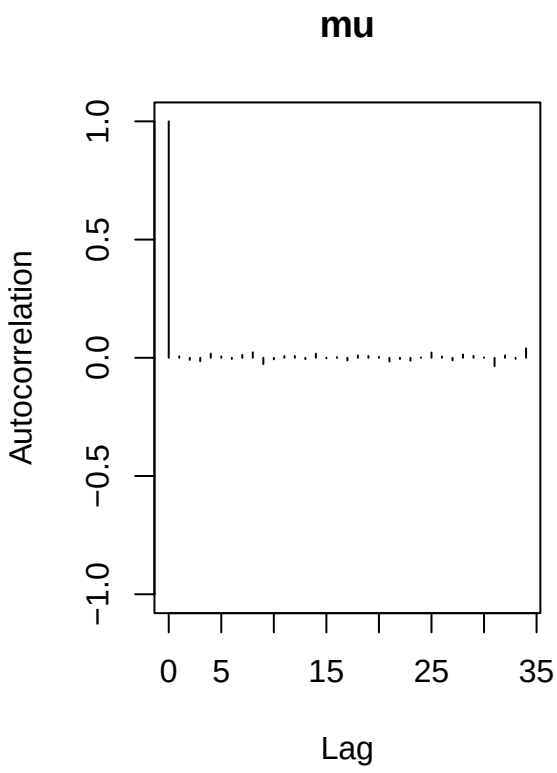
```
coda::autocorr.plot(draws_multiple_chains)
```

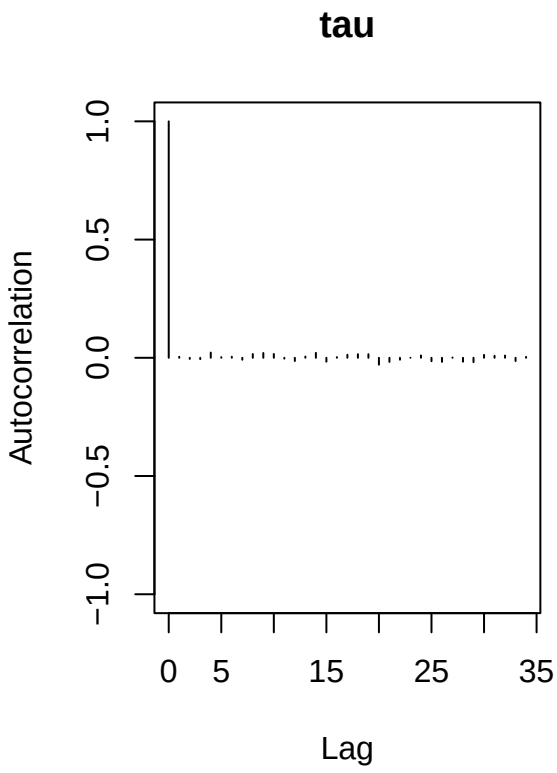

mu
Autocorrelation
1.0
0.5
0.0
−0.5
−1.0
0
5
15
25
35
Lag

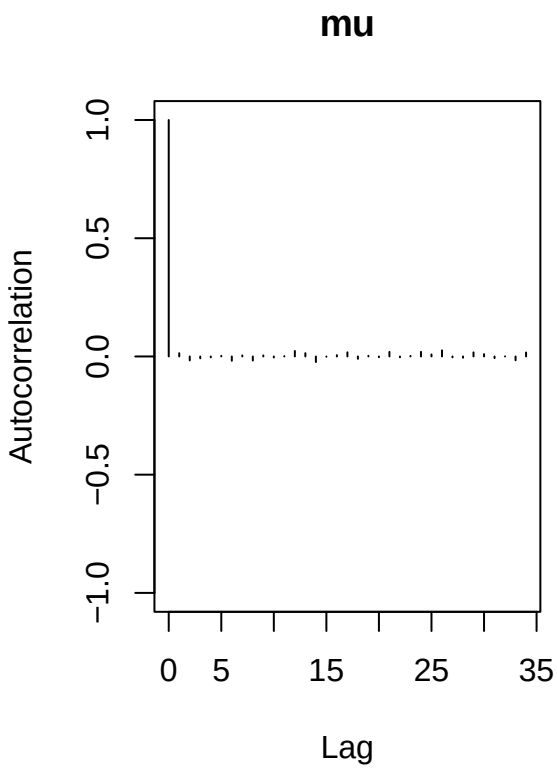

tau
Autocorrelation
1.0
0.5
0.0
−0.5
−1.0
0
5
15
25
35
Lag

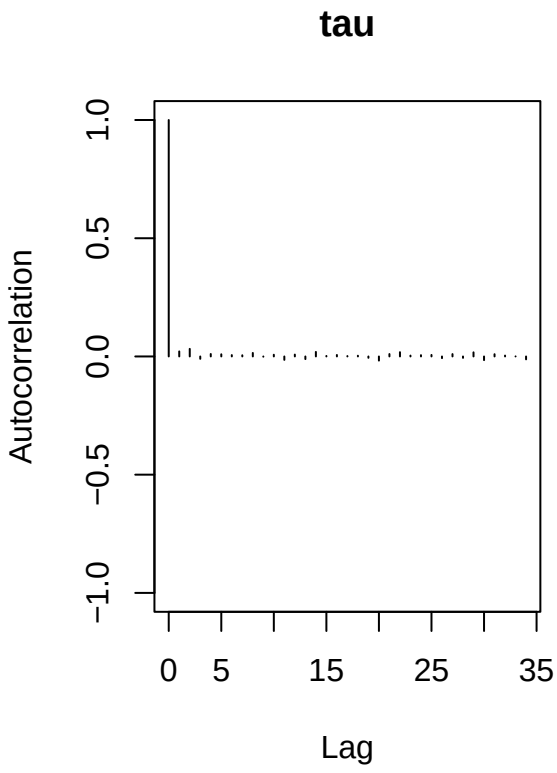

mu
Autocorrelation
1.0
0.5
0.0
−0.5
−1.0
0
5
15
25
35
Lag


tau
Autocorrelation
1.0
0.5
0.0
−0.5
−1.0
0
5
15
25
35
Lag


mu
Autocorrelation
1.0
0.5
0.0
−0.5
−1.0
0
5
15
25
35
Lag


tau
Autocorrelation
1.0
0.5
0.0
−0.5
−1.0
0
5
15
25
35
Lag


mu
Autocorrelation
1.0
0.5
0.0
−0.5
−1.0
0
5
15
25
35
Lag


tau
Autocorrelation
1.0
0.5
0.0
−0.5
−1.0
0
5
15
25
35
Lag