More MCMC diagnostics

Stat 340: Bayesian Statistics

Tuning MCMC

Three main decisions:

- Selecting the initial values for the parameters
- Determining if/when the chain(s) has converged
- Selecting the number of samples needed to approximate the posterior

Initial values

- The algorithm will eventually converge no matter what initial values you select
- Choosing good starting values will speed up convergence
- It is important to try a few initial values to verify they all give the same result
- Usually 3-5 separate chains is sufficient

Common strategies:

1. Select good initial values using method of moments or MLE
2. Purposely pick bad but different initial values for each chain to check convergence

Specifying initial values

Specify the initial conditions for the MCMC chain using the `inits` argument

For a single chain, pass in a named list with your starting values

```
# Pass this to inits argument
init_list <- list(mu = 1, phi = 1/5)
```

For multiple chains, pass in a list of named lists with different starting values

```
# Pass this to inits argument
init_list2 <- list(
  list(mu = 1, phi = 1/5),
  list(mu = -1, phi = 1/10)
)
```

Specifying initial values

You can also use random starting values if you don't have a good guess, or if you model is quite complicated

```
inits_gen <- function(){
  # function for initializing random mu, phi starting values
  list(mu = rnorm(1, 0, 100), phi = 1 / runif(1, 0.001, 1000))
}
```

But remember, poor starting values can require longer burn-in periods

A note on reproducibility

Our usual approach to reproducile simulations is to run `set.seed(1234)` (with a better seed), but this won't work with JAGS

Instead, specify the `.RNG.seed` and `.RNG.name` in the `inits` argument

```
repro_inits <- list(
  list(.RNG.name = "base::Wichmann-Hill", .RNG.seed = 2019,
       mu = rnorm(1, 0, 100), phi = 1 / runif(1, 0.001, 1000)),
  list(.RNG.name = "base::Wichmann-Hill", .RNG.seed = 2020,
       mu = rnorm(1, 0, 100), phi = 1 / runif(1, 0.001, 1000)),
  list(.RNG.name = "base::Wichmann-Hill", .RNG.seed = 2021,
       mu = rnorm(1, 0, 100), phi = 1 / runif(1, 0.001, 1000))
)
```

A note on reproducibility

There are four psuedorandom number generators in base JAGS:

- `"base::Wichmann-Hill"`
- `"base::Marsaglia-Multicarry"`
- `"base::Super-Duper"`
- `"base::Mersenne-Twister"`

R uses these as well, so you can read more at `?Random`

FWIW, `"base::Mersenne-Twister"` if the default in R

Convergence diagnostics

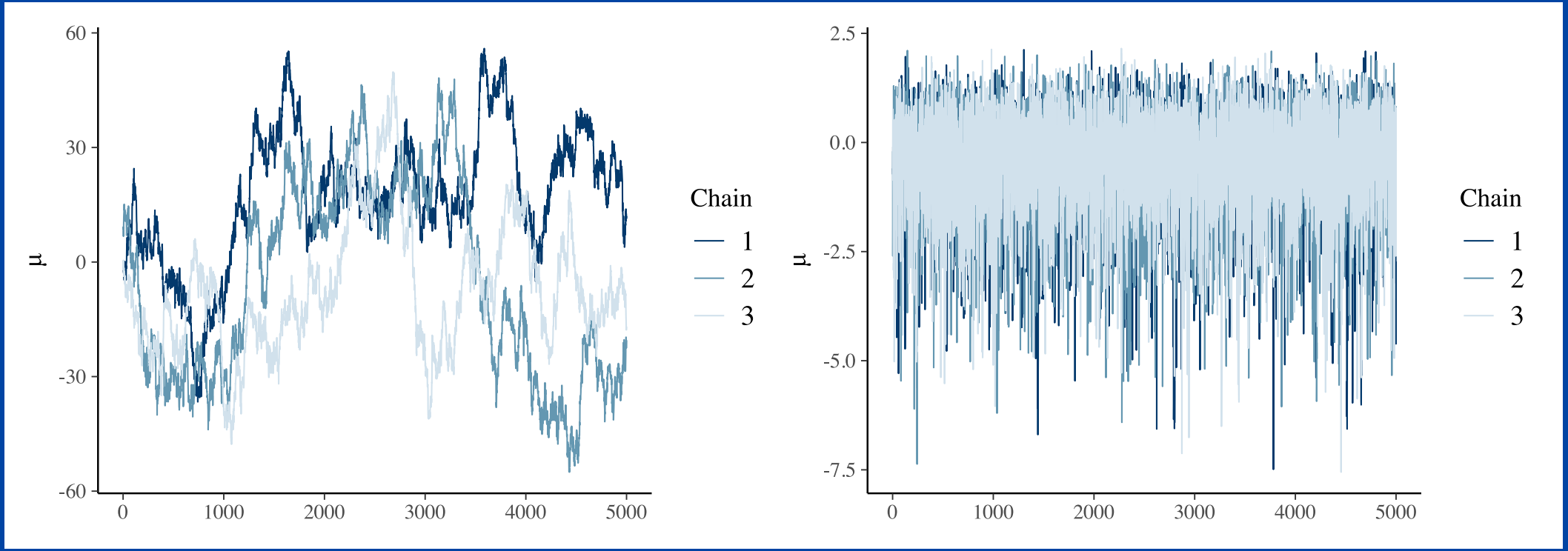
> Don't assume that your Markov chain converged to the desired stationary distribution just because "it worked."

A **"good" Markov chain** should be:

- **Stationary**: draws should within the posterior (stable path around the center of gravity)
- **Well mixed**:
  - successive draws should not be highly correlated
  - each chain should target the same stationary distribution

Your turn

With a neighbor, decide whether each chain is well mixed



Tools already in our toolkit

Did my chains converge?

- Trace plots

Are my chains well mixed?

- ACF plots

Convergence diagnostics

The **coda** R package has dozens of diagnostics

```
library(coda)
```

Did my chains converge?

- Geweke
- Gelman-Rubin

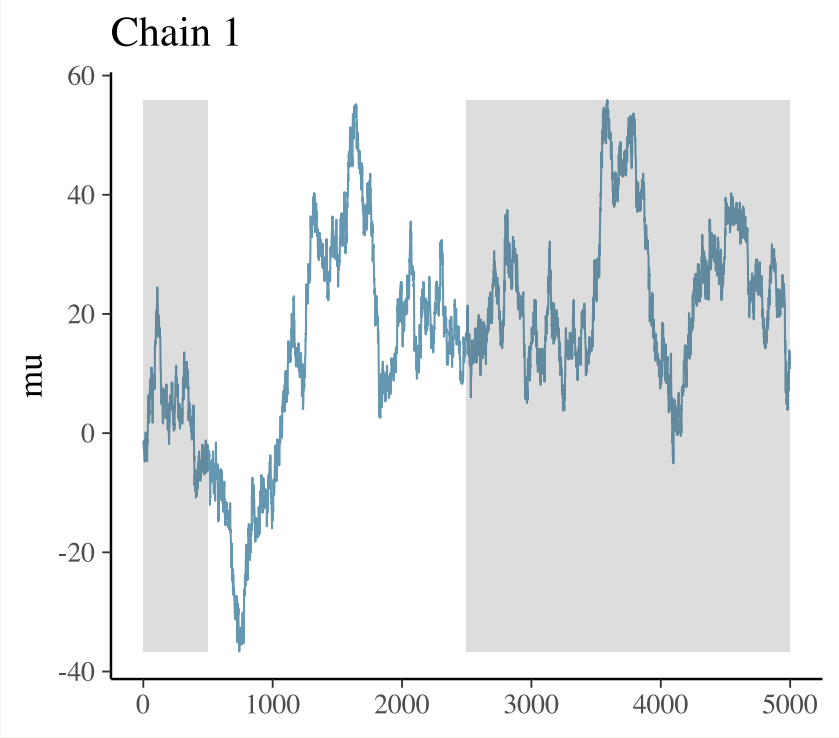
Did I run the sampler long enough after convergence?

- Effective sample size
- Standard errors for the posterior mean estimate

Geweke diagnostic

- **Idea:** a two-sample test comparing the mean of a chain between batches at the beginning versus the end
- By default, JAGS compares the first 10% with the last 50%
- Test statistic is a Z-score with the standard errors adjusted for autocorrelation (so we won't write down the formula)

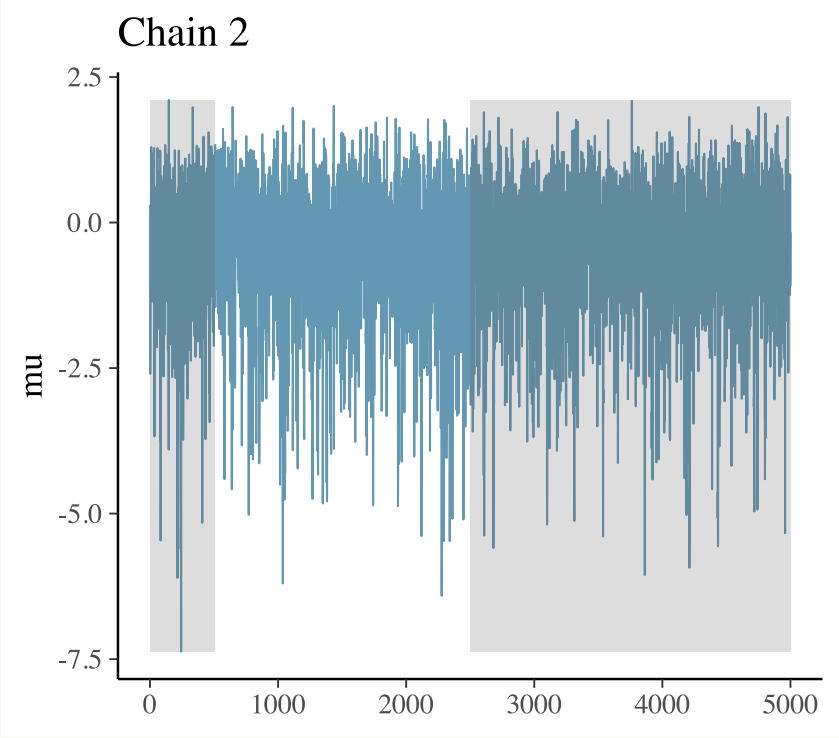
Geweke diagnostic




```
geweke.diag(bad_samples$mcmc[[1]])
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##     mu
## -3.646
```

Geweke diagnostic



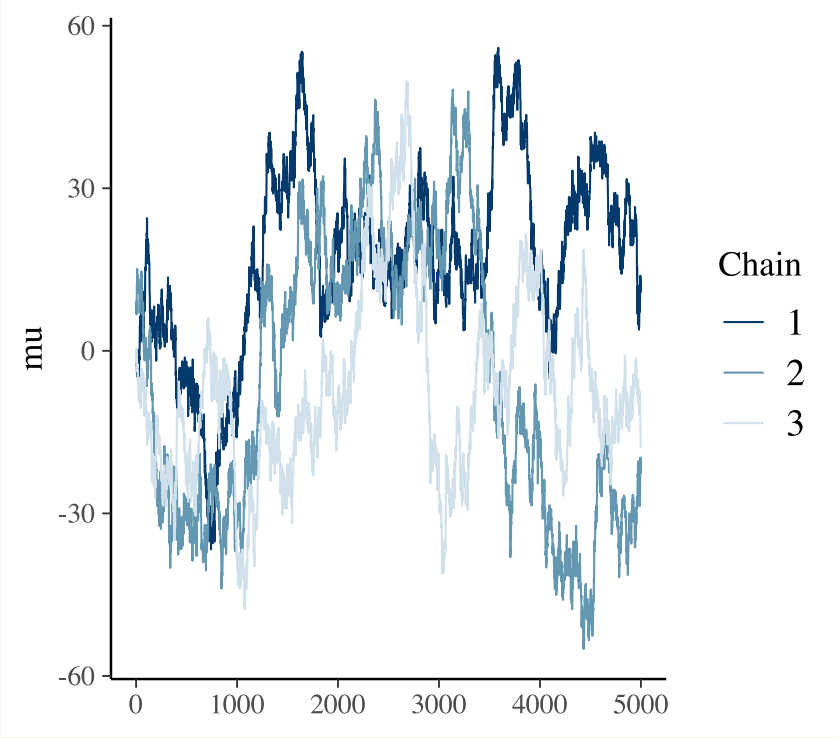

```
geweke.diag(good_samples$mcmc[[2]])
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##      mu
## -0.2094
```

Gelman-Rubin diagnostic

- Effective convergence of Markov chain simulation has been reached when inferences for quantities of interest do not depend on the starting point of the simulations.
- If we run multiple chains, we hope that all chains give same result
- Gelman and Rubin (1992) proposed (essentially) an ANOVA test of whether the chains have the same mean
- R_j is scaled and approaches 1 from above
  - $R_j = 1 \Rightarrow$ perfect convergence
  - $R_j \geq 1.1 \Rightarrow$ red flag

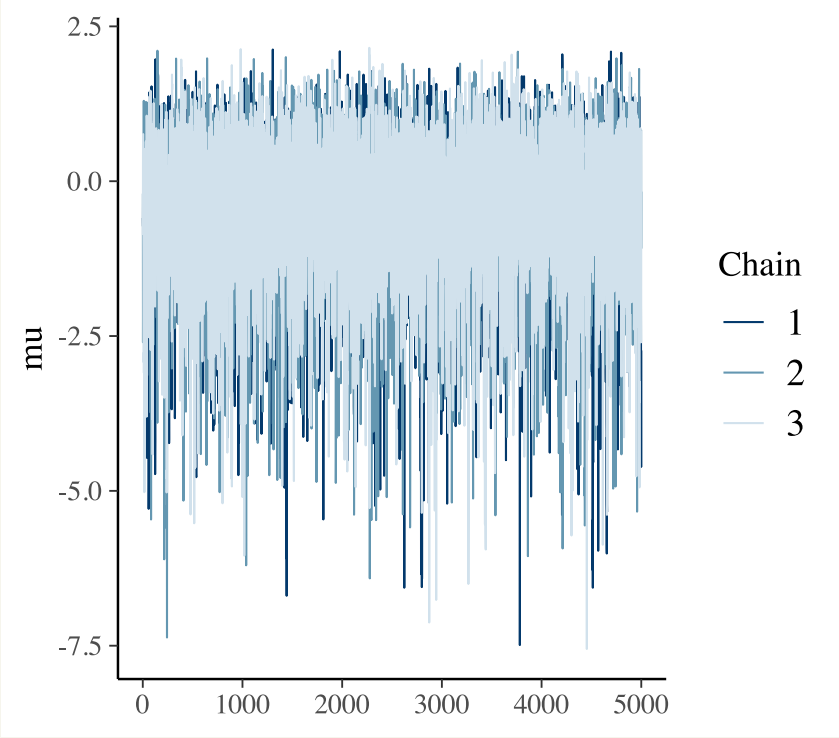
Gelman-Rubin diagnostic




```
gelman.diag(bad_samples$mcmc)
```

```
## Potential scale reduction factors
##
##    Point est. Upper C.I.
## mu       1.41       2.38
```

Gelman-Rubin diagnostic




```
gelman.diag(good_samples$mcmc)
```

```
## Potential scale reduction factors
##
##    Point est. Upper C.I.
## mu          1          1
```

A note on autocorrelation

- Lower values are better, but if the chains are long enough even large values can be OK
- **Thinning** the Markov chain means keeping only every kth draw, where k is chosen so that the autocorrelation is small
- `thin` argument to `run.jags()` implements this
- This is always less efficient than using all samples, but can save memory

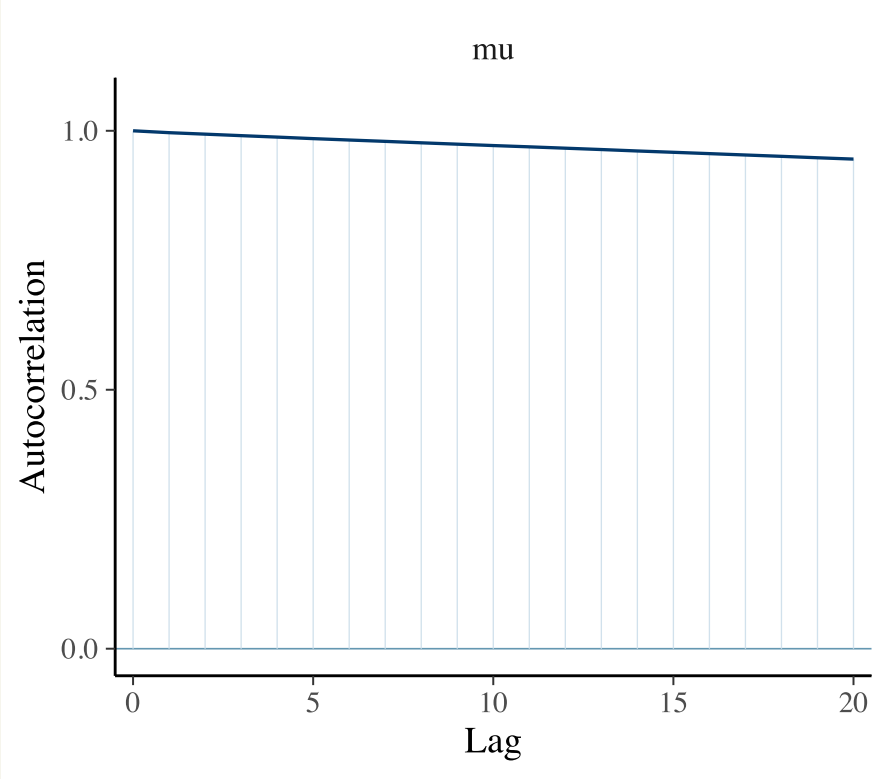
Effective sample size

- Highly correlated samples have less information than independent samples
- $S =$ # MCMC samples after burn in
- **Effective samples size (ESS)**

$$ESS_k = S \Big/ \left(1 + 2\sum_{k=1}^{\infty} \rho(k)\right)$$

- ESS = "equivalent number of independent observations"
- Should be at least a few thousand for all parameters

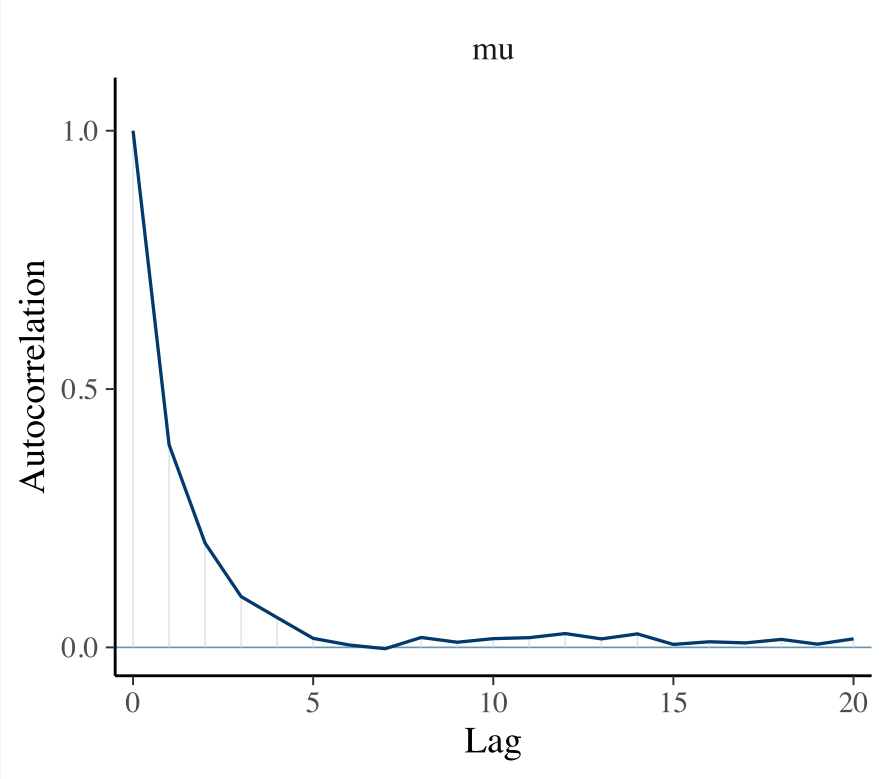
Effective sample size




```
# ESS for single chain of mu1
# n.iter = 5000
effectiveSize(bad_samples$mcmc[[1]])
```

```
##       mu
## 7.016737
```

Effective sample size




```
# ESS for single chain of mu1
# n.iter = 5000
effectiveSize(good_samples$mcmc[[1]])
```

```
##        mu
## 1947.605
```

Standard errors of posterior mean estimates

- Assuming independence the standard error is

$$\text{Naive SE} = \frac{s}{\sqrt{S}}$$

  where s = sample SD

- A more realistic standard error is

$$\text{Time-series SE} = \frac{s}{\sqrt{ESS}}$$

- If the SE is too large, rerun the MCMC algorithm for a larger number of samples

Standard errors of posterior mean estimates

```
summary(bad_samples$mcmc)
##
## Iterations = 2001:7000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##         1.7099        23.1165          0.1887         5.4610
##
## 2. Quantiles for each variable:
##
##    2.5%     25%     50%     75%   97.5%
## -39.955 -16.786   3.175  19.307  43.773
```

Standard errors of posterior mean estimates

```
summary(good_samples$mcmc)
##
## Iterations = 2001:7000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##       -0.56735        1.26785         0.01035        0.01592
##
## 2. Quantiles for each variable:
##
##    2.5%     25%     50%     75%   97.5%
## -3.6671 -1.2308 -0.3638  0.3374  1.3089
```

What to do if the chains haven't converged?

- Increase the number of iterations
- Tune the Metropolis candidate distribution
- Use better initial values
- Use a more advanced algorithm
- Simplify/reparameterize the model
- Use (more) informative priors