STA 360/602L: Module 3.10

MCMC and Gibbs sampling IV

Dr. Olanrewaju Michael Akande

Some MCMC terminology

- Convergence: bypassing initial drift in the samples towards a stationary distribution.
- Burn-in: samples at start of the chain that are discarded to allow convergence.
- Trace plot: plot of sampled values of a parameter vs iterations.
- Slow mixing: tendency for high autocorrelation in the samples.
- Thinning: practice of collecting every kth iteration to reduce autocorrelation.

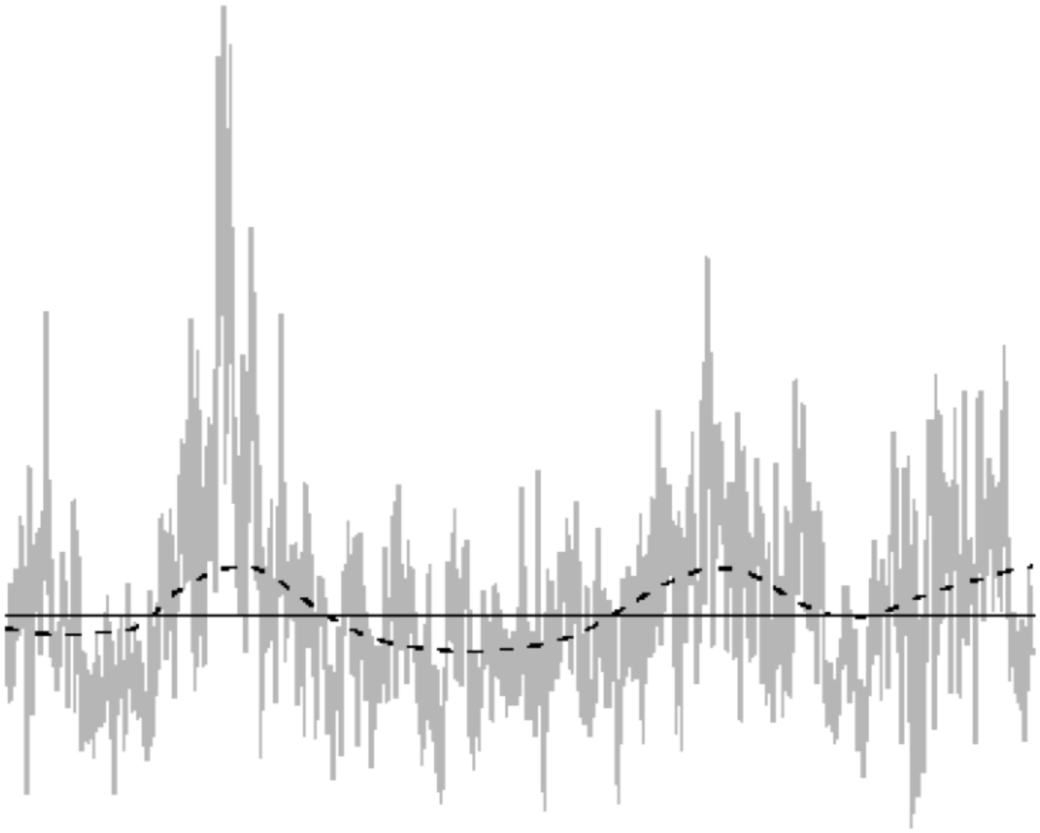
  It gets you a little closer to iid draws and saves memory (you don't store all draws), but unless memory is a major issue or autocorrelation is very high, thinning is usually not needed.

Burn-in

- Because convergence often occurs regardless of our starting point (in not-too-complex problems at least), we can usually pick any reasonable values in the parameter space as a starting point.
- The time it takes for the chain to converge may vary depending on how close the starting values are to a high probability region of the posterior.
- Generally, we throw out a certain number of the first draws, known as the **burn-in**, as an attempt to make our draws closer to the stationary distribution and less dependent on any single set of starting values.
- However, we don't know exactly when convergence occurs, so it is not always clear how much burn-in we would need.

Trace plot with bad mixing

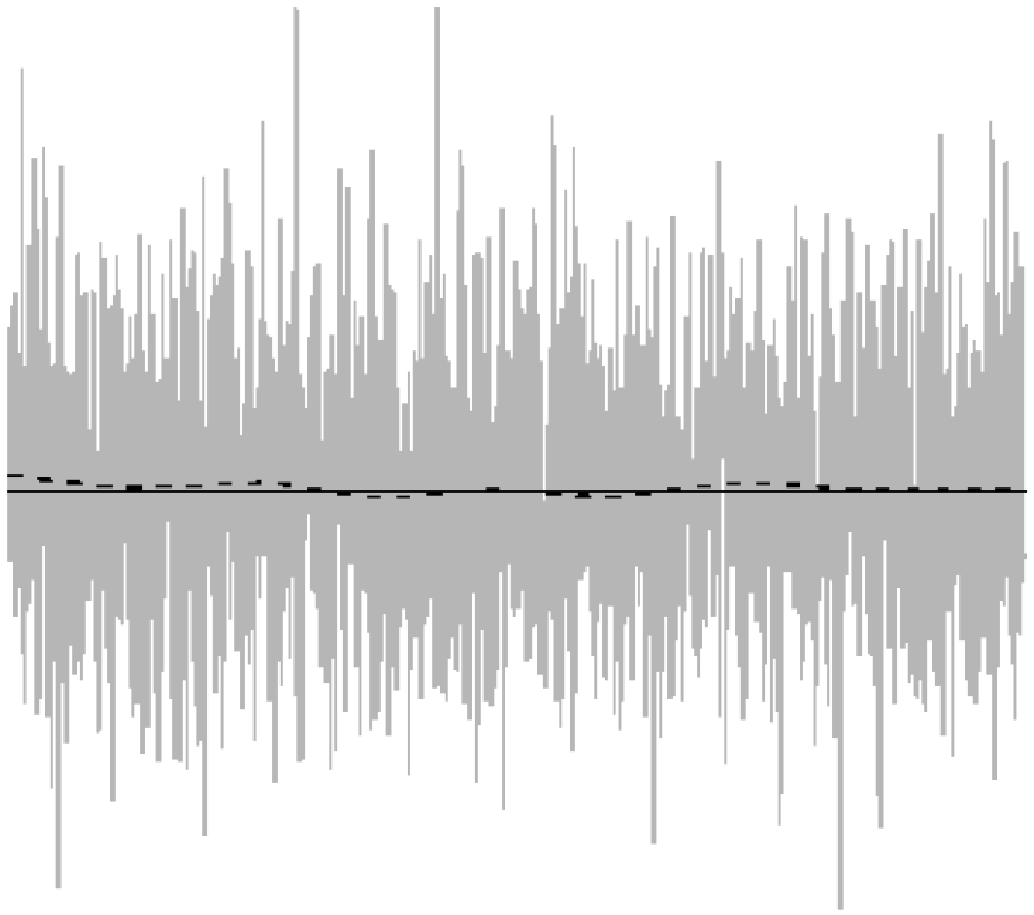
- Trace plot: plot of sampled values of a parameter vs iterations.



Poor mixing

- Exhibits "snaking" behavior in trace plot with cyclic local trends in the mean.
- Poor mixing in the Gibbs sampler caused by high posterior correlation in the parameters.
- Decreases efficiency & many more samples need to be collected to maintain low Monte Carlo error in posterior summaries.
- For very poor mixing chain, may even need millions of iterations.
- Routinely examine trace plots!

Trace plot with good mixing



Convergence diagnostics

- Diagnostics available to help decide on number of burn-in & collected samples.
- **Note**: no definitive tests of convergence but you should do as many diagnostics as you can, on all parameters in your model.
- With "experience", visual inspection of trace plots perhaps most useful approach.
- There are a number of useful automated tests in R.

Diagnostics in R

- The most popular package for MCMC diagnostics in R is `coda`.
- `coda` uses a special MCMC format so you must always convert your posterior matrix into an MCMC object.
- Continuing with the posterior samples for the Pygmalion study, we have the following in R.

```r
#library(coda)
phi.mcmc <- mcmc(PHI,start=1) #no burn-in (simple problem!)
```

Diagnostics in R

```
summary(phi.mcmc)
```

```
##
## Iterations = 1:10000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean       SD  Naive SE Time-series SE
## mu     13.98961  2.94748 0.0294748      0.0341435
## tau     0.02839  0.01646 0.0001646      0.0001855
## sigma2 53.34388 53.27616 0.5327616      0.6502608
##
## 2. Quantiles for each variable:
##
##             2.5%      25%      50%      75%     97.5%
## mu      7.519819 12.36326 14.21682 15.84203  19.27701
## tau     0.005744  0.01626  0.02526  0.03726   0.06886
## sigma2 14.522591 26.83933 39.59569 61.49382 174.10833
```

The naive SE is the **standard error of the mean**, which captures simulation error of the mean rather than the posterior uncertainty.

The time-series SE adjusts the naive SE for **autocorrelation**.

Effective sample size

- The effective sample size translates the number of MCMC samples S into an equivalent number of independent samples.
- It is defined as

$$\mathrm{ESS} = \frac{S}{1 + 2\sum_k \rho_k},$$

where S is the sample size and ρ_k is the lag k autocorrelation.

- For our data, we have

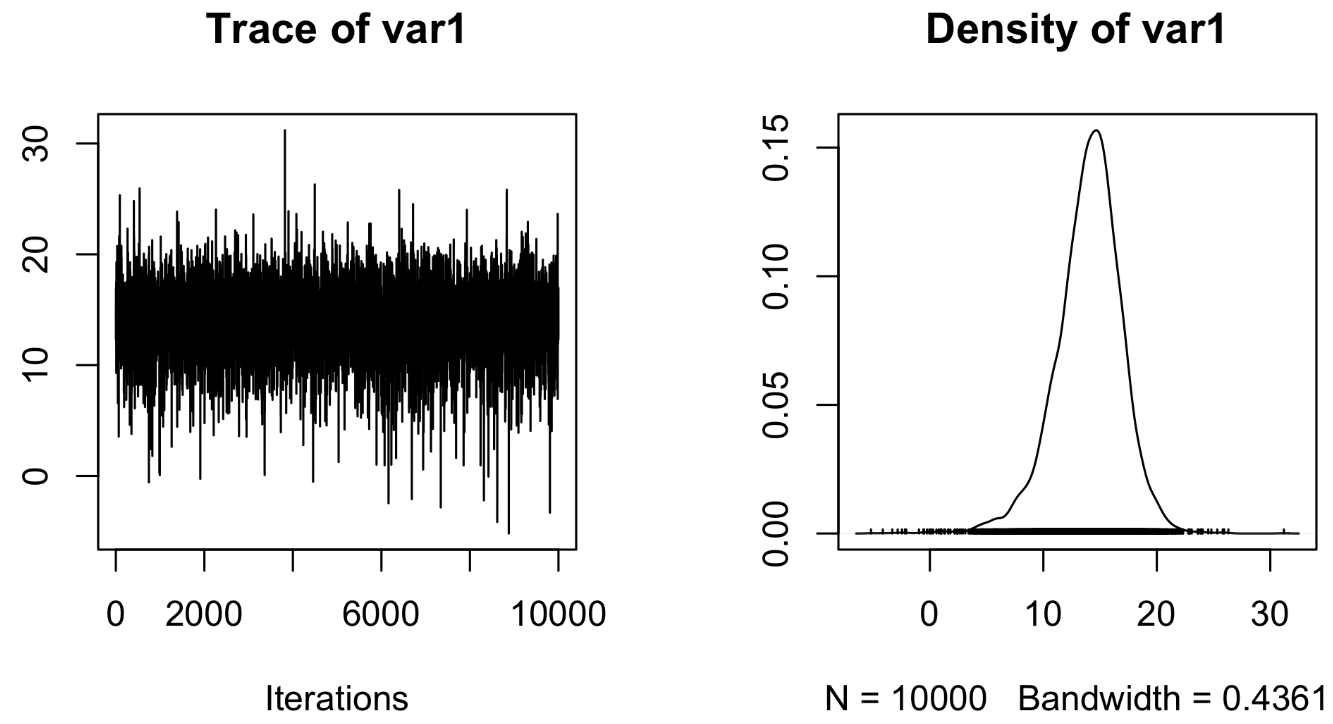
```
effectiveSize(phi.mcmc)
```

```
##       mu      tau   sigma2
## 7452.197 7877.721 6712.600
```

- So our 10,000 samples are equivalent to 7452 independent samples for μ, 7878 independent samples for τ, and 6713 independent samples for σ^2.

Trace plot for mean

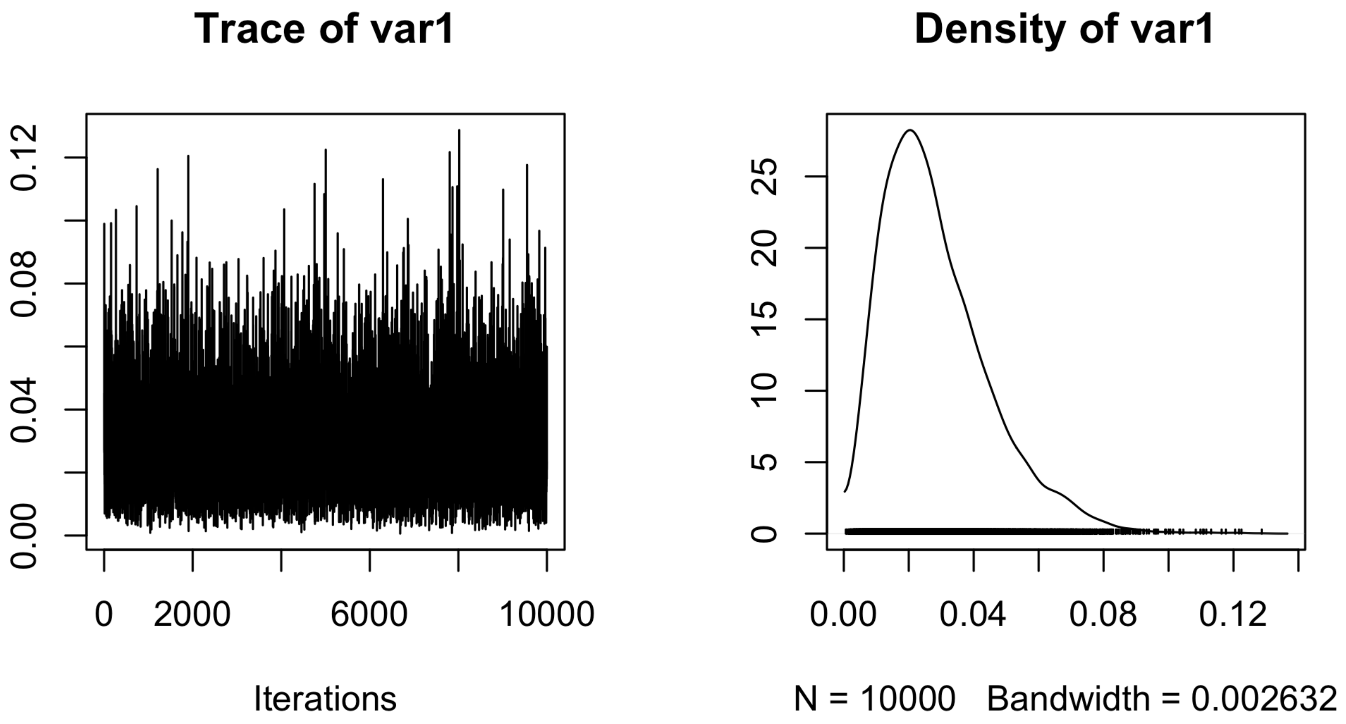
```
plot(phi.mcmc[,"mu"])
```



Looks great!

Trace plot for precision

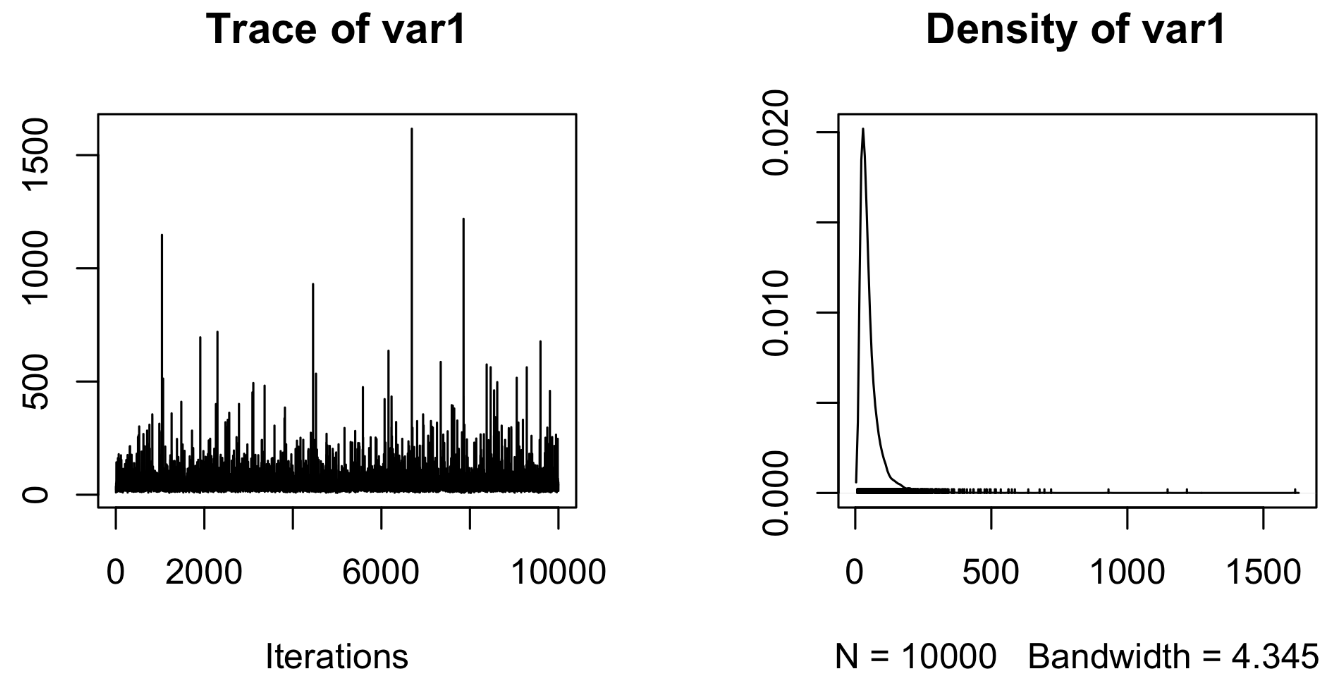
```
plot(phi.mcmc[,"tau"])
```



Looks great!

Trace plot for variance

```
plot(phi.mcmc[,"sigma2"])
```



We do see a few wacky samples that we did not see with τ, due to the scale. Generally, still looks great!

Autocorrelation

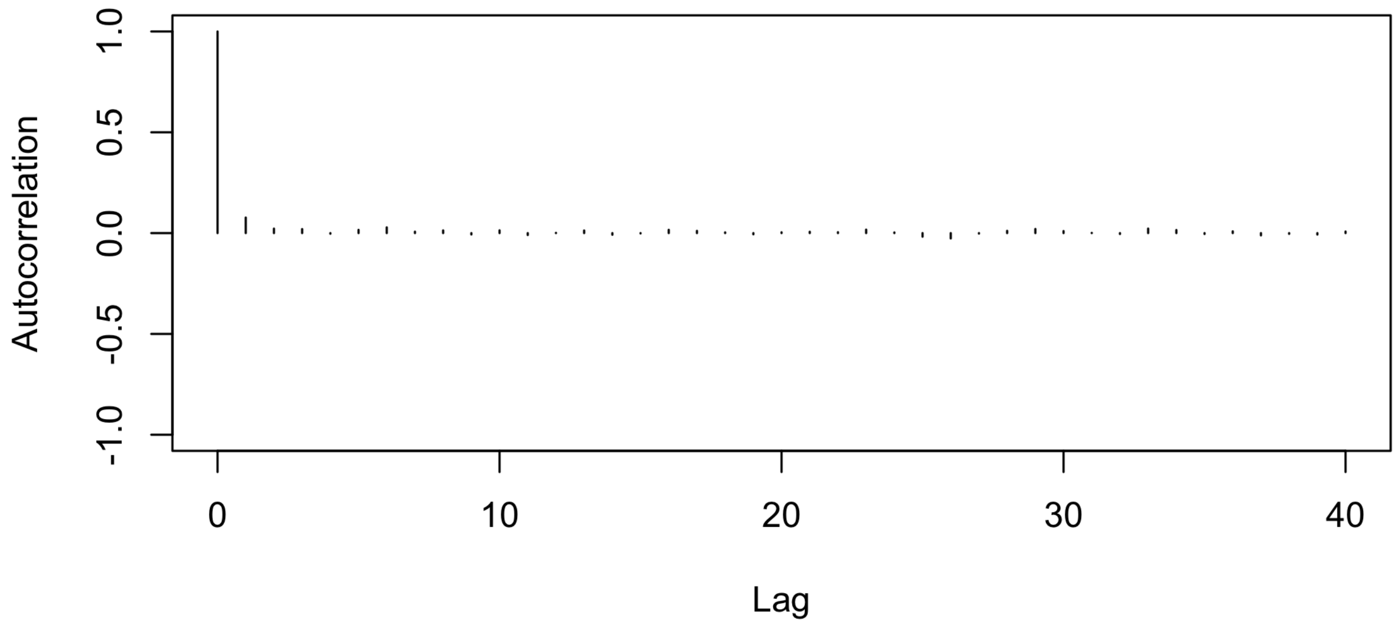
- Another way to evaluate convergence is to look at the autocorrelation between draws of our Markov chain.
- The lag k autocorrelation, ρ_k, is the correlation between each draw and its kth lag, defined as

$$\rho_k = \frac{\sum_{s=1}^{S-k}(\theta_s - \bar{\theta})(\theta_{s+k} - \bar{\theta})}{\sum_{s=1}^{S-k}(\theta_s - \bar{\theta})^2}.$$

- We expect the autocorrelation to decrease as k increases.
- If autocorrelation remains high as k increases, we have slow mixing due to the inability of the sampler to move around the space well.

Autocorrelation for mean

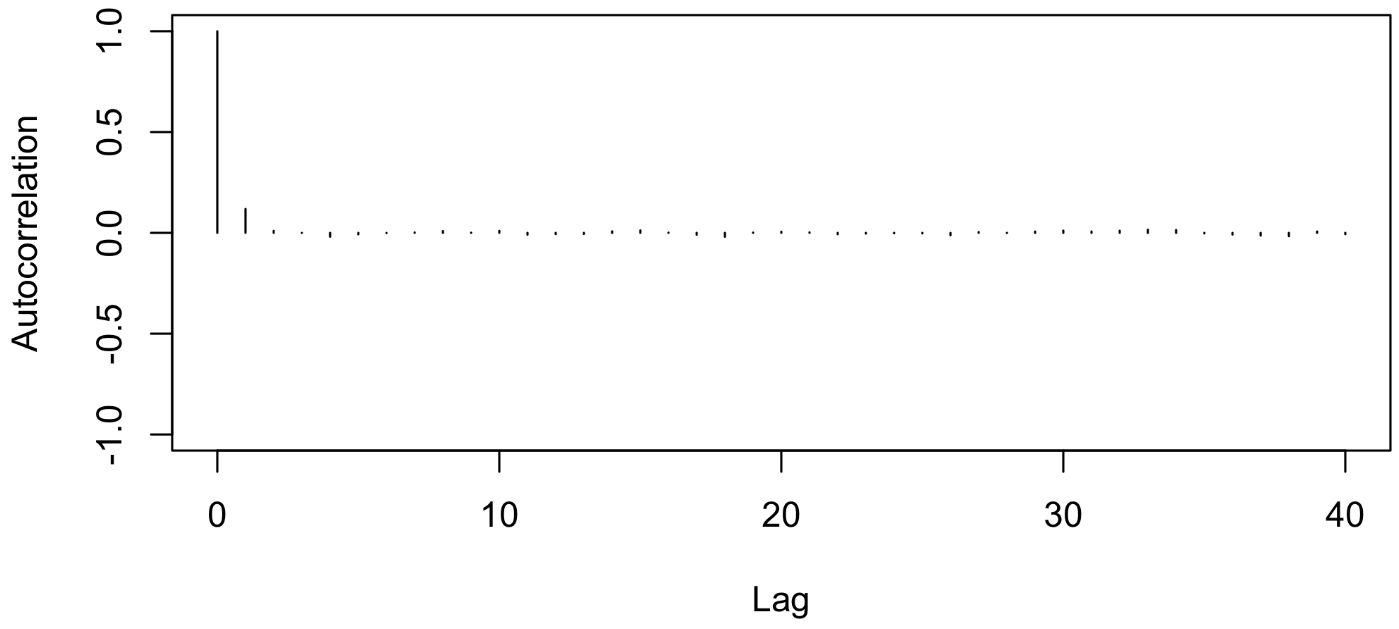
```
autocorr.plot(phi.mcmc[,"mu"])
```



This looks great! Look how quickly autocorrelation goes to 0.

Autocorrelation for precision

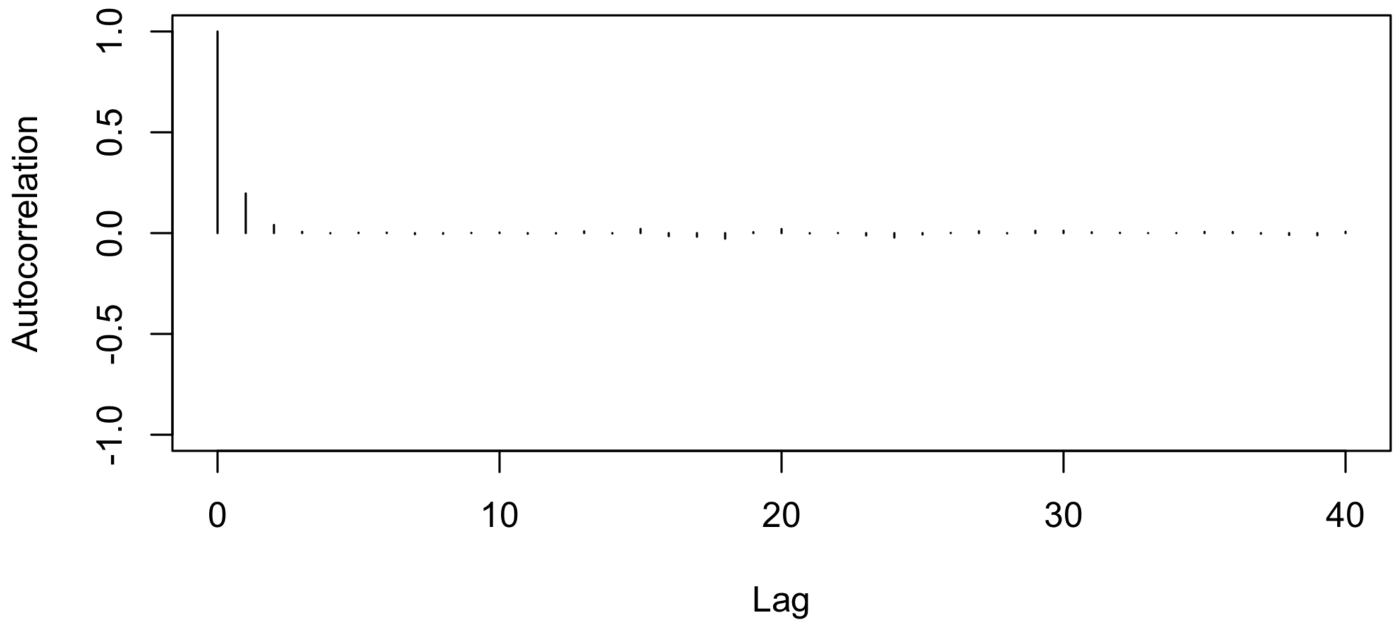
```
autocorr.plot(phi.mcmc[,"tau"])
```



Also great!

Autocorrelation for variance

```r
autocorr.plot(phi.mcmc[,"sigma2"])
```



Also great!

Gelman and Rubin Statistic

- Andrew Gelman and Don Rubin suggested a diagnostic statistic based on taking separate sets of Gibbs samples (multiple chains) with dispersed initial values to test convergence.

- The algorithm proceeds as follows.
  - Run m > 2 chains of length 2S from overdispersed starting values.
  - Discard the first S draws in each chain.
  - Calculate the within-chain and between-chain variance.
  - Calculate the estimated variance of the parameter as a weighted sum of the within-chain and between-chain variance.
  - Calculate the potential scale reduction factor

$$\hat{R} = \sqrt{\frac{\hat{\mathrm{Var}}(\theta)}{W}},$$

where $\hat{\mathrm{Var}}(\theta)$ is the weighted sum of the within-chain and between-chain variance and W is the mean of the variances of each chain (average within-chain variance).

Geweke statistic

- Geweke proposed taking two non-overlapping parts of a single Markov chain (usually the first 10% and the last 50%) and comparing the mean of both parts, using a difference of means test.
- The null hypothesis would be that the two parts of the chain are from the same distribution.
- The test statistic is a z-score with standard errors adjusted for autocorrelation, and if the p-value is significant for a variable, you need more draws.
- The output is the z-score itself (not the p-value).

```
geweke.diag(phi.mcmc)
```

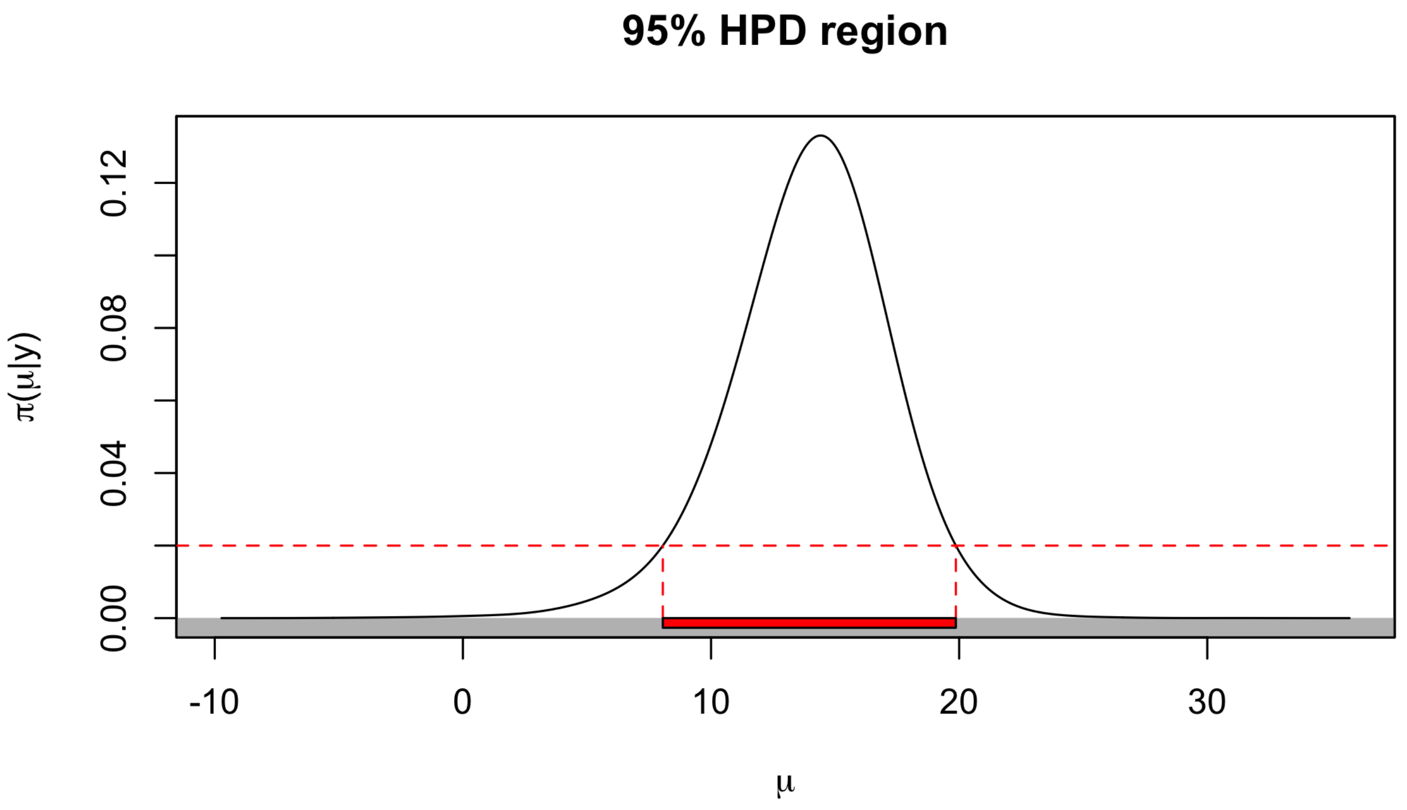
```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##      mu     tau  sigma2
##  0.9521  2.0088 -1.9533
```

Practical advice on diagnostics

- There are more tests we can use: Raftery and Lewis diagnostic, Heidelberger and Welch, etc.
- The Gelman-Rubin approach is quite appealing in using multiple chains
- Geweke (and Heidelberger and Welch) sometimes reject even when the trace plots look good.
- Overly sensitive to minor departures from stationarity that do not impact inferences.
- Sometimes this can be solved with more iterations. Otherwise, you may want to try multiple chains.
- Most common method of assessing convergence is visual examination of trace plots.
- **CAUTION**: diagnostics cannot guarantee that a chain has converged, but they can indicate it has not converged.

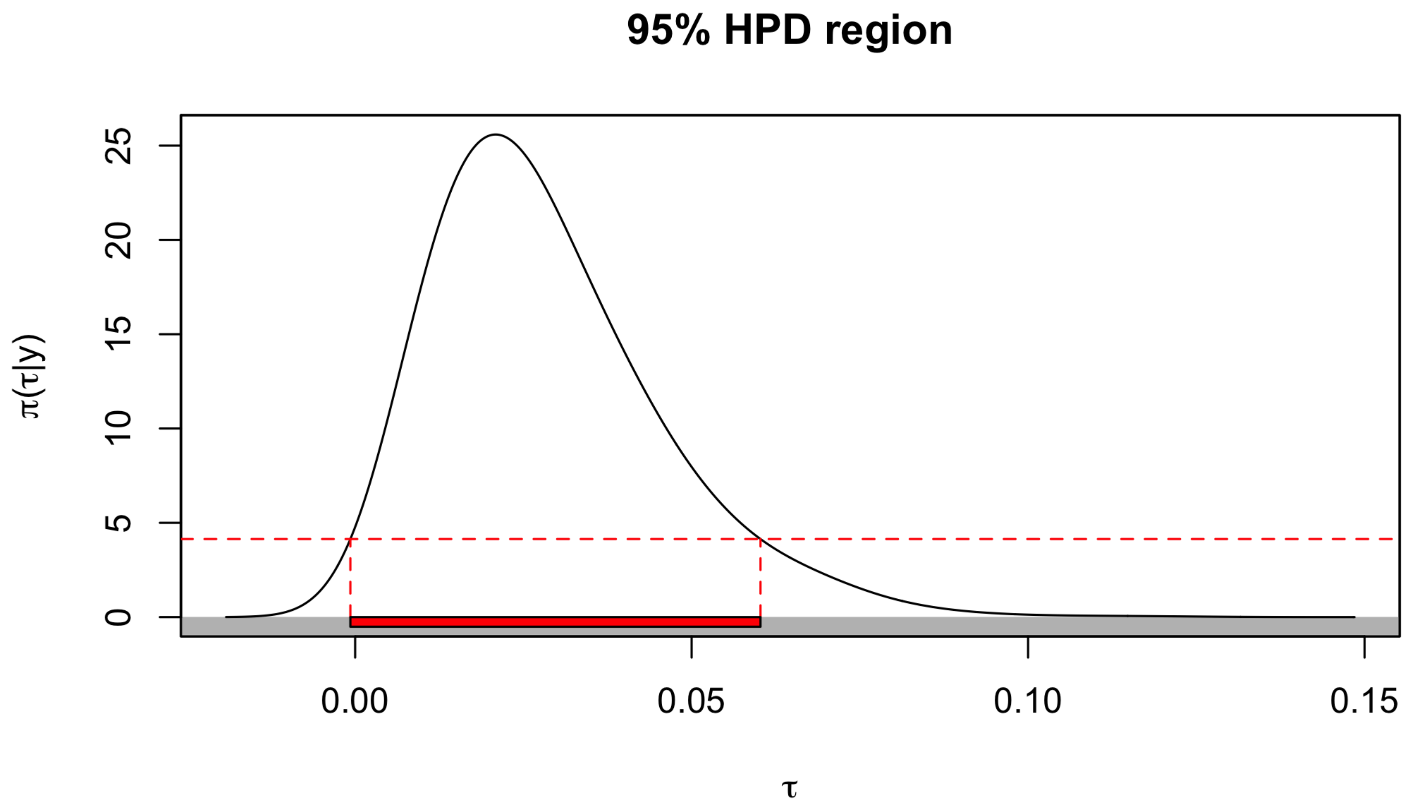
HPD interval for Pygmalion data

```r
#library(hdrcde)
hdr.den(PHI[,1],prob=95,main="95% HPD region", xlab=expression(mu),
        ylab=expression(paste(pi,"(", mu, "|y)")))
```



HPD interval for Pygmalion data

```r
hdr.den(PHI[,2],prob=95,main="95% HPD region", xlab=expression(tau),
        ylab=expression(paste(pi,"(", tau, "|y)")))
```



HPD interval for Pygmalion data

```
hdr(PHI[,1],prob=95)$hdr
```

```
##         [,1]     [,2]
## 95% 8.080022 19.87699
```

```
hdr(PHI[,2],prob=95)$hdr
```

```
##              [,1]       [,2]
## 95% -0.0006954123 0.06023567
```

We can compare the HPD intervals to the equal tailed credible intervals.

```
quantile(PHI[,1],c(0.025,0.975))
```

```
##      2.5%     97.5%
##  7.519819 19.277013
```

```
quantile(PHI[,2],c(0.025,0.975))
```

```
##        2.5%       97.5%
## 0.005743552 0.068858238
```

Intervals are closer for μ (symmetric density) compared to τ (not symmetric).

What's next?

Move on to the readings for the next module!