STATS 230: Computational Statistics
Convergence of Markov Chain Monte Carlo

Babak Shahbaba

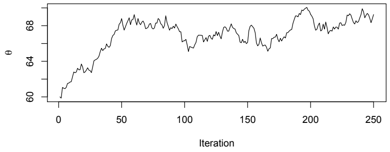
Department of Statistics, UCI

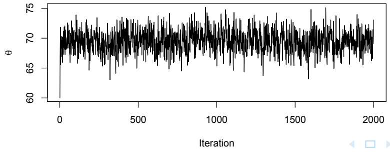
Convergence diagnosis

- When using MCMC methods for sampling, it is important to make sure that the chain has converged to its target distribution. That is, it has reached its stationary phase.
- Furthermore, we need to make sure that enough samples have been generated after convergence to provide a good approximation.
- There are many formal convergence diagnostic methods which basically ensure that there is no overall upward or downward trends in the sequence of samples (they would go up and down locally of course), and the chain would converge to the same distribution regardless of its initial point.

Convergence diagnosis

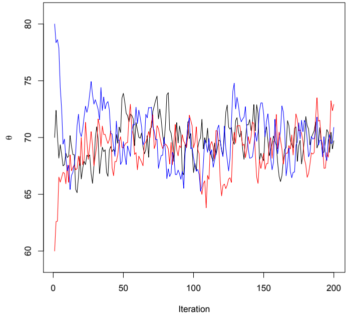
- In many cases, we can simply evaluate convergence based on the trace plot of MCMC samples.
- Here, the top graph shows that the chain has not converged yet after 200 iterations since it is still moving down.
- The bottom graph shows that the chain has eventually converged.





Convergence diagnosis

- It is common to discard MCMC samples prior to the convergence of the chain.
- In this case, we say we *burn-in* pre-convergence samples.
- It is also recommended to run multiple chains from different starting points.
- In the following graph, three chains with three different starting points converge to the same distribution.

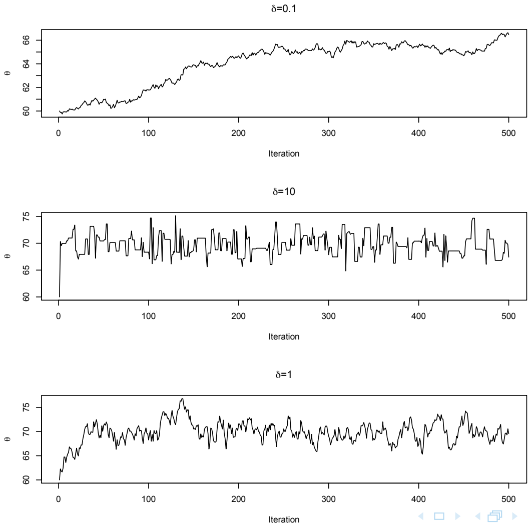


Step-size effect

- in many MCMC algorithms, the speed of convergence is controlled by some parameters, which can be tuned (we call these *tuning parameters*).
- In Metropolis and MH, for example, the step size (δ) of the proposal distribution is a tuning parameter.
- If we use small steps (e.g., small δ in Uniform$(x - \delta, x + \delta)$ or $N(x, \delta^2)$ proposal distributions) the Markov chain might take a long time to converge. If we use large steps, the acceptance rate might be very low leading to inefficiency.

Step-size effect

- The following graph shows the effect of step size on Metropolis and its convergence.



Step-size effect

- We need to find δ such that the chain has a good acceptance rate (usually around 40% for simple Metropolis with univariate proposal and around 20% when we are updating large number of parameters at the same time) and converges fast.
- We can, for example, run the chain for few iterations and find the acceptance rate. Reduce the step sizes if the acceptance rate is too low, or increase it if the autocorrelation is high (i.e., the chain is moving very slowly), and start the chain again.
- CAUTION: You cannot fine tune the step size during the run based on the statistic (e.g., acceptance rate) you collect from previous iterations since this undermine the Markov property.
- Read the Short-Cut MCMC method by Radford Neal.

Using coda library in R for diagnosis

- After you obtain your MCMC samples (possibly from multiple chains), you can perform convergence diagnosis using the `coda` package in R.
- For the univariate normal example with known variance, I ran three chains each with 20000 iterations.
- The samples from these three chains are stored in vectors `theta1, theta2`, and `theta3`.
- We start with one chain, `theta1`, for now. We need to convert this vector to an MCMC object:

```
> theta1 <- mcmc(theta1)
```

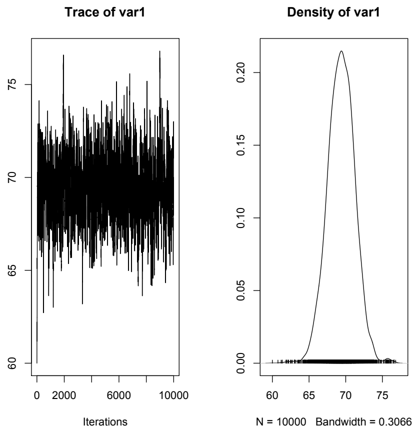
Summarizing the posterior distribution

- You can then obtain the summary statistics

```
> summary(theta1)

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
      69.37378        1.84386        0.01844        0.07950

2. Quantiles for each variable:

 2.5%   25%   50%   75% 97.5%
65.78 68.16 69.40 70.60 72.89
```

Plotting the posterior distribution

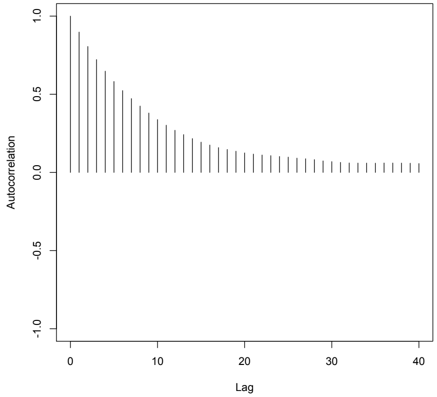
- You can also plot the the samples and the density of the posterior distribution using `plot(theta1)`



- Alternatively, you can plot them separately using `traceplot(theta)` and `densplot(theta)`.

Autocorrelation

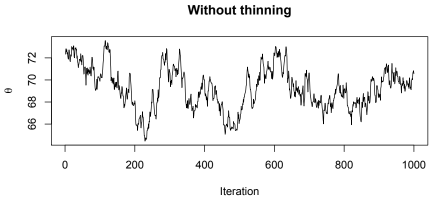
- Now let's start our diagnosis by plotting the autocorrelation function for different lags using `autocorr.plot(theta1)`

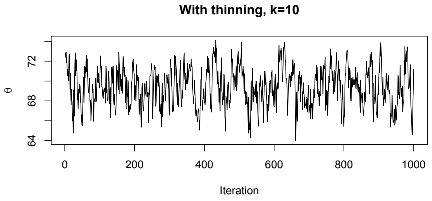


- For a good chain, autocorrelation diminishes quickly as the lags increase.

Thinning

- A quick fix to reduce autocorrelation is *thinning*: we include (store) only every k^{th} (e.g., 10^{th}) sample.





- This of course reduces the sample size. A more efficient solution would be to find a better step size.

Geweke's method

- A simple convergence diagnosis test was proposed by Geweke (1992).
- In this approach, we test equality of the means for the first segment (usually the first 10% of samples) and the last segment (usually the last 50% of samples) of a Markov chain.
- The test statistic is a standard z-score, which is the difference between the two sample means divided by its estimated standard error taking into account any autocorrelation.
- If the samples are drawn from the stationary distribution of the chain, then the two means are expected to be equal.
- In this case, Geweke's statistic has an asymptotically standard normal distribution.

Geweke's method

- To perform Geweke's test, we can use the following function:

```
> geweke.diag(theta1)

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

  var1
-0.198
```

- By default, the function compares the first 10% and the last 50% of MCMC samples.
- The z-score in this case is very small.

Gelman and Rubin' method

- Another criteria for evaluating convergence is to estimate *within* chain variance and *between* chains variance. When chains converge, these two variances should be close.
- Let's assume we run m independent chains each of size n (after discarding the ones prior to convergence).
- Denote the i^{th} sample in the j^{th} chain as θ'_{ij}.
- Between variance is then

$$B = \frac{n}{m-1} \sum_{j=1}^{m} (\overline{\theta'}_{.j} - \overline{\theta'}_{..})^2$$

$$\overline{\theta'}_{.j} = \frac{1}{n} \sum_{i=1}^{n} \theta'_{ij} \qquad \overline{\theta'}_{..} = \frac{1}{m} \sum_{j=1}^{m} \theta'_{.j}$$

- Within variance is

$$W = \frac{1}{m} \sum_{j=1}^{m} s_j^2 \qquad \text{where } s_j^2 = \frac{1}{n-1} \sum_{i=1}^{n} (\theta'_{ij} - \overline{\theta'}_{.j})^2$$

Gelman and Rubin' method

- The variance of $\theta|y$ is then estimated based on W and B

$$\widehat{Var}^{+}(\theta|y) = \frac{n-1}{n}W + \frac{1}{n}B$$

- When a chain reaches its stationary phase, the expectation of W reaches the true value of $Var(\theta|y)$. In general W is less than the true value.

- We can stop the chains when following quantity is close to 1 (e.g., less than 1.1) and reduces only marginally as n increases (Gelman and Rubin):

$$\hat{R} = \sqrt{\frac{\widehat{Var}^{+}(\theta|y)}{W}}$$

Gelman and Rubin' method

- We can use coda to find R.
- First, we need to create a new `mcmc` object that includes all three chains:

```
> theta1 <- mcmc(theta1)
> theta2 <- mcmc(theta2)
> theta3 <- mcmc(theta3)
> theta.ls <- mcmc.list(theta1, theta2, theta3)
```

- Now we can perform Gelman and Rubin's test using the following function

```
> gelman.diag(theta.ls)

Potential scale reduction factors:

     Point est. Upper C.I.
[1,]       1.01       1.02
```

- In the above example, $\hat{R} = 1.01$.

Monte Carlo with dependent samples

- After we make sure the chain has converged, we discard (burn-in) all the pre-convergence samples, and use the remaining samples for the Monte Carlo estimation.
- There is, however, one issue we need to address.
- Recall that Monte Carlo requires independent samples (after all, it's based on the law of large numbers).
- We know the samples obtained from MCMC are not independent.
- However, for long chains, both the law of large numbers ($\frac{1}{nm} \sum h(\theta') \rightarrow E[h(\theta)]$) and CLT components of the Monte Carlo method still hold. The only difference is that due to autocorrelation of samples, the variance $\sigma*^2$ of $N(0, \sigma*^2)$ from CLT would be bigger than what it would have been if the *nm* samples where completely independent (see Neal 1993 for detail).

Monte Carlo with dependent samples

- We can, of course, use every k^{th} draw (thinning) to reduce autocorrelation.
- By doing this, however, we are also reducing the sample size.
- Alternatively, we could use all the samples, but remember that the effective number of independent samples (*effective sample size*) is

$$n_{\text{eff}} = \min[nm, nm\frac{\widehat{Var}^{+}(\theta|y)}{B}]$$

- We can find the effective sample size as follows:

```
>  effectiveSize(theta1)

    var1
537.9448
```

Autocorrelation time

- We can evaluate the efficiency of an MCMC method based on its *autocorrelation time*, which is defined as the number of iterations required to generate an independent sample.
- We can estimate the autocorrelation time (ACT) by dividing the n posterior samples batches of size B and estimating ACT as follows (Neal, 1993, and Geyer, 1992):

$$\tau = B\frac{S_b^2}{S^2}$$

- Here, S^2 is the sample variance and S_b^2 is the sample variance of batch means.

Some general comments

- Always evaluate your codes on simulated data or models with closed form for which you know the answer.
- Always try multiple starting points to make sure the chains are converging to the same distribution and they are not trapped in a subspace.
- Always perform convergence diagnosis; don't assume that since your chain has been running for a long time it's guaranteed to converge.
- Don't undermine Markov chain properties required for convergence to the target distribution. For example, don't reduce the step size (i.e., changing the transition distribution) at the middle of a run if the acceptance rate has been low for the past few iterations. There are better ways (e.g., slice sampling, Short-Cut MCMC) for tuning the step size.