Bayesian Data Analysis

Vanda Inácio & Ken Newman

University of Edinburgh

Semester 2, 2017/2018

Bayesian computing

- As we've seen, Bayesian inference centres around the posterior distribution

$$p(\boldsymbol{\theta} \mid \mathbf{y}) = \frac{f(\mathbf{y} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})}{\int f(\mathbf{y} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}} \propto f(\mathbf{y} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}).$$

- Bayesian inference thus requires summarising the posterior distribution.
- When there are more than a few parameters, this requires advanced computational tools.
- We will briefly discuss some methods that are deterministic. These either apply only for low-dimensional problems or rely on asymptotic approximations:
  - Bayesian central limit theorem.
  - Numerical integration.
- Monte Carlo methods are more general:
  - Direct sampling.
  - Markov chain Monte Carlo methods.

Bayesian computing

Bayesian central limit theorem

- The classical approach often computes the mle, and then assumes (based on asymptotic theory) that its sampling distribution is Gaussian for inference.

- The Bayesian central limit theorem can be used in the same way to summarise a posterior.

- **Bayesian central limit theorem**: Suppose $Y_1, \ldots, Y_n \overset{\text{iid}}{\sim} f(\mid \theta)$ and that the prior $p(\theta)$ and the likelihood $f(\mathbf{y} \mid \theta)$ are positive and twice differentiable near $\widehat{\theta}_{\text{post}}$, the posterior mode of θ. Then for large n

$$p(\theta \mid \mathbf{y}) \sim \text{N}\left(\widehat{\theta}_{\text{post}}, [I^{\text{post}}(\theta, \mathbf{y})]^{-1}\right),$$

where

$$I^{\text{post}}(\theta, \mathbf{y}) = -\left[\frac{\partial^2}{\partial\theta\partial\theta^T} \log p(\theta \mid \mathbf{y})\right]_{\theta=\widehat{\theta}_{\text{post}}}.$$

Bayesian computing

Bayesian central limit theorem

- Other forms of the normal approximation are occasionally used.
- For instance, if the prior is reasonably flat, we might ignore it in the previous calculation.
- This in effect replaces the posterior mode $\widehat{\boldsymbol{\theta}}_{\text{post}}$ by the mle $\widehat{\boldsymbol{\theta}}$ and $I^{\text{post}}(\boldsymbol{\theta}, \mathbf{y})$ is the usual observed Fisher information matrix

$$I(\boldsymbol{\theta}, \mathbf{y}) = -\left[\frac{\partial^2}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^T} \log f(\mathbf{y} \mid \boldsymbol{\theta})\right]_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}}.$$

- The use of the Bayes CLT in practice has diminished over the past few years, due to concerns about the quality of the approximation combined with the increasing ease of exact solutions implemented via MCMC.

Bayesian computing

Numerical integration

- The vast majority of posterior summaries are integrals of the posterior (posterior mean, variance, probability above zero, etc).
- There is a vast literature on approximating integrals.
- These work great in medium dimensions, say models with 5-10 parameters.
- The simplest method is a grid approximation. Suppose we have an unnormalised posterior distribution we want to sample from. The grid approximation works as follows:
  1. Divide the unnormalised posterior area into m grids.
  2. Evaluate each grid point at the unnormalised posterior.
  3. Sample grid points with unnormalised posterior ordinates as the probabilities.

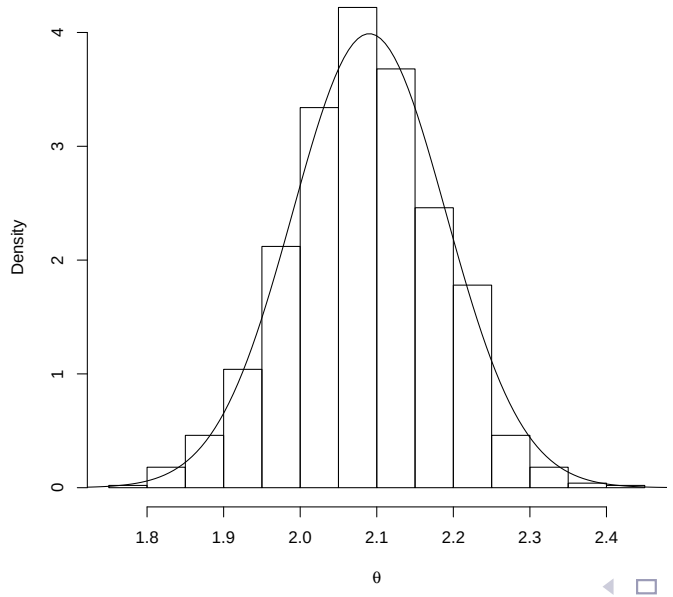
Bayesian computing

Numerical integration

- Suppose $y_1, \ldots, y_n$ is observed data and that $Y_i \overset{\text{iid}}{\sim} \text{N}(\theta, 1)$, and $\theta \sim \text{N}(\mu_0 = 0, \sigma_0^2 = 10^2)$.
- We do not need numerical integration here since we do know that $p(\theta \mid \mathbf{y})$ is actually available in closed form, namely

$$\theta \mid \mathbf{y}, \sigma^2 \sim \text{N}\left(\frac{\frac{\mu_0}{\sigma_0^2} + \frac{n\bar{\mathbf{y}}}{\sigma^2}}{\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}}, \frac{1}{\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}} \right).$$

- This is only a toy example to illustrate the use of the method.



Bayesian computing

Numerical integration

- ↪ Implementation available on the `R` script `example_grid_approximation`.
- ↪ There are obviously better approximation methods. The best is probably INLA:

`http://www.r-inla.org/`

Bayesian computing

Monte Carlo sampling

- Monte Carlo sampling is the predominant method of Bayesian inference because it avoids asymptotic approximations and can be used in high-dimensions.
- The main idea is to approximate posterior summaries by drawing samples from the posterior distribution, and then using these samples to approximate posterior summaries of interest.
- For example, if $\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(S)}$ are samples from $p(\boldsymbol{\theta} \mid \mathbf{y})$, then the mean of S samples can be used to approximate the posterior mean.
- This only provides approximations of the posterior summaries of interest.
- Many argue that this form of approximation is superior to asymptotic approximations because the Bayes CLT requires the sample size of the dataset to go to infinity and the Monte Carlo approximation requires the number of simulated values to go to infinity.
- In most cases, $S \to \infty$ is cheaper and more realistic than $n \to \infty$.
- But how to draw samples from some arbitrary distribution $p(\boldsymbol{\theta} \mid \mathbf{y})$?

Bayesian computing

Monte Carlo sampling: Rejection sampling

- Suppose we wish to generate values $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(S)}$ from the posterior $p(\theta \mid \mathbf{y})$ but this is not a known distribution.
- Rejection sampling takes samples from a distribution that resembles the posterior and is easy to sample from (say a normal approximation using the CLT), and thins those samples to obtain draws from the posterior distribution.
- The approximate density, $g(\theta)$, is called the envelope function.
- Let M be a constant so that $p(\theta \mid \mathbf{y}) \leq Mg(\theta)$ for all θ. Then $p(\theta \mid \mathbf{y})$ resides in the envelope.

Bayesian computing

Monte Carlo sampling: Rejection sampling

↪ The rejection sampling algorithm is:

1. Sample $\theta \sim g(\theta)$ and draw $u \sim \text{Unif}(0, 1)$.
2. Set $\alpha = \frac{p(\theta|\mathbf{y})}{Mg(\theta)}$.
3. If $u \leq \alpha$, accept θ.

Repeat until you have accepted S (pre-determined) values. The accepted values $\theta^{(1)}, \ldots, \theta^{(S)}$ are a random sample from $p(\theta \mid \mathbf{y})$.

Bayesian computing

Monte Carlo sampling: Rejection sampling

- ↪ It is sometimes tricky to choose a good envelope/proposal distribution $g(\theta)$. A basic requirement is that it should have support at least as large as $p(\theta \mid \mathbf{y})$ and preferably heavier tails than $p(\theta \mid \mathbf{y})$.

- ↪ It is desirable to choose M as small as possible for efficiency reasons. Note that $M \geq \frac{p(\theta|\mathbf{y})}{g(\theta)}$ for all θ, so that the optimal M is simply

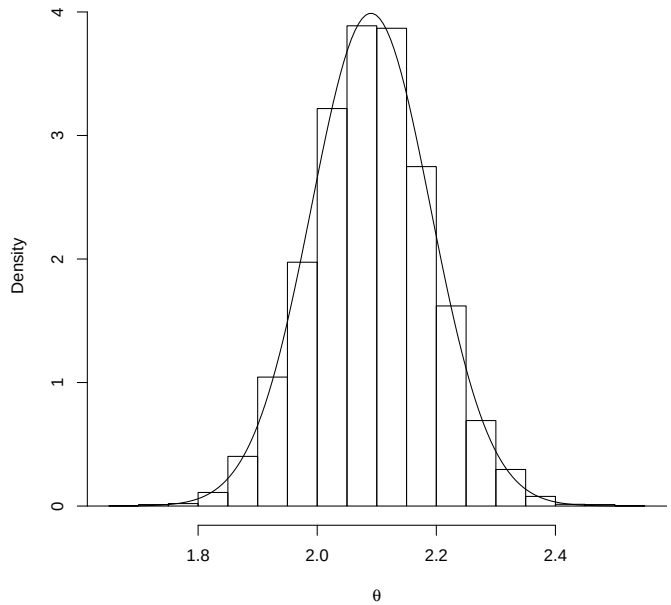
$$M^{\text{opt}} = \max_{\theta} \left(\frac{p(\theta \mid \mathbf{y})}{g(\theta)} \right).$$

- ↪ Note that rejection sampling can be extended to the case where the normalisation constant is unknown. This works because we can essentially incorporate the normalisation constant into the M term.

Bayesian computing

Monte Carlo sampling: Rejection sampling

- We return to the toy example used to illustrate the grid approximation to the posterior.
- In this context, we choose as an envelope distribution, a normal distribution with mean equal to the mean of the data and standard deviation equal to the standard deviation of the data.



- Implementation available on the `R` script `example_rejection_sampling`.

Bayesian computing

Monte Carlo sampling: Importance sampling

- We once more wish to obtain a sample from the posterior distribution $p(\theta \mid \mathbf{y})$ which we assume is difficult to do directly.
- Suppose that we can easily sample from other distribution $g(\theta)$. As with rejection sampling, we require g to have, at least, the same support as the posterior distribution.
- Suppose further that we are interested in obtaining an estimate of $\mathbb{E}_{\text{post}}[f(\theta)]$ (e.g., the posterior mean).
- While rejection sampling makes an absolute decision to either accept or reject a candidate, importance sampling, in turn, gives partial credit.
- Let $\theta^{(1)}, \ldots, \theta^{(S)}$ be samples from g and define importance weights

$$\omega(\theta^{(s)}) = \frac{p(\theta^{(s)} \mid \mathbf{y})}{g(\theta^{(s)})}.$$

- Then we can estimate $\mathbb{E}_{\text{post}}[f(\theta)]$ by

$$\widehat{\theta}_g = \frac{1}{S} \sum_{s=1}^{S} \omega(\theta^{(s)}) f(\theta^{(s)}).$$

Bayesian computing

Monte Carlo sampling: Importance sampling

↪ The method also works when the normalisation constant is unknown using

$$\widehat{\theta}_g = \frac{\sum_{s=1}^{S} \omega(\theta^{(s)}) f(\theta^{(s)})}{\sum_{s=1}^{S} \omega(\theta^{(s)})}.$$

↪ In the `R` script, `example_importance_sapling` we illustrate the implementation of the method to compute the posterior mean for our toy example.

Bayesian computing

Monte Carlo sampling: Sampling importance resampling

- The sampling importance resampling (SIR) algorithm is in its simplest form is a simple extension of the importance sample algorithm.

- The algorithm is as follows:

1. Draw values $\theta^{(1)}, \ldots, \theta^{(S)}$ from $g(\theta)$.
2. Calculate the importance weights

$$\omega(\theta^{(s)}) = \frac{p(\theta^{(s)} \mid \mathbf{y})}{g(\theta^{(s)})}, \quad s = 1, \ldots, S.$$

3. Normalise the weights

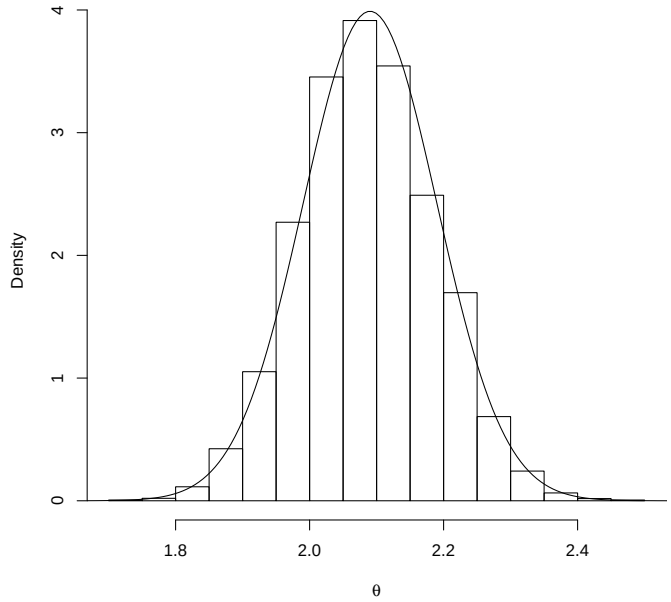
$$\omega^*(\theta^{(s)}) = \frac{\omega(\theta^{(s)})}{\sum_{s=1}^{S} \omega(\theta^{(s)})}, \quad s = 1, \ldots, S..$$

4. Resample with replacement from $\{\theta^{(1)}, \ldots, \theta^{(S)}\}$ where the probability of sampling $\theta^{(s)}$ is given by $\omega^*(\theta^{(s)})$.

Bayesian computing

Monte Carlo sampling: Sampling importance resampling

- ↪ This scheme works also if $p(\theta \mid \mathbf{y})$ is known up to a proportionality constant (because the normalisation constant cancels out in step 3).
- ↪ Samples from g which 'fits best to $p(\theta \mid \mathbf{y})$' are those most likely to appear in the resample. However, if g is a poor approximation to the posterior these will not necessarily be good samples (in the sense of resembling the posterior).
- ↪ We apply the SIR algorithm to our toy example. See details on the `R` script `example_sir`.



Bayesian computing

Monte Carlo sampling

- ↪ All these three direct sampling methods (rejection sampling, importance sampling, and sampling importance resampling) suffer from the problem of dimensionality.
- ↪ These methods can be generally implemented to obtain posterior estimates of summary statistics of interest in one dimension but become significantly more difficult (and generally impossible) to implement efficiently in higher dimensions.
- ↪ More general methods are needed.

Bayesian computing

MCMC

- MCMC techniques are based on the construction of a Markov chain that eventually 'converges' to the target distribution (called stationary or equilibrium) which, in our case, is the posterior distribution $p(\boldsymbol{\theta} \mid \mathbf{y})$.
- This is the main way to distinguish MCMC algorithms from direct simulation methods, which provide samples directly from the target- posterior distribution.
- Moreover, the MCMC output is a dependent sample since it is generated from a Markov chain, in contrast to the output of direct methods, which is an independent sample.
- Finally, MCMC methods incorporate the notion of an iterative procedure (for this reason they are frequently called iterative methods) since in every step they produce values depending on the previous one.
- We will briefly cover two different MCMC methods: Gibbs sampler and Metropolis-Hastings.

Bayesian computing

MCMC: Gibbs sampler

- Gibbs sampling was proposed in the early 1990s (Geman and Geman, 1984; Gelfand and Smith, 1990) and fundamentally changed Bayesian computing.
- Gibbs sampling is attractive because it can sample from high-dimensional posteriors.
- The main idea is to break the problem of sampling from the high-dimensional joint distribution into a series of samples from low-dimensional conditional distributions.
- The algorithm begins by setting initial values for all parameters, $\boldsymbol{\theta}^{(0)} = (\theta_1^{(0)}, \ldots, \theta_p^{(0)})$.
- Variables are then samples one at a time from their full conditional distributions

$$p(\theta_j \mid \theta_1, \ldots, \theta_{j-1}, \theta_{j+1}, \ldots, \theta_p, \mathbf{y}), \qquad j = 1, \ldots, p.$$

- Rather than 1 sample from p-dimensional joint, we make p-dimensional samples.
- The process is repeated until the required number of samples have been generated.

Bayesian computing

MCMC: Gibbs sampler

↪ Formally, the algorithm is:

1. Set initial values $\boldsymbol{\theta}^{(0)} = (\theta_1^{(0)}, \ldots, \theta_p^{(0)})$.
2. For $s = 1, \ldots, S$:
   - Draw $\theta_1^{(s)} \sim p(\theta_1 \mid \theta_2^{(s-1)}, \theta_3^{(s-1)}, \ldots, \theta_p^{(s-1)}, \mathbf{y})$.
   - Draw $\theta_2^{(s)} \sim p(\theta_2 \mid \theta_1^{(s)}, \theta_3^{(s-1)}, \ldots, \theta_p^{(s-1)}, \mathbf{y})$.
   - ...
   - Draw $\theta_p^{(s)} \sim p(\theta_p \mid \theta_1^{(s)}, \theta_2^{(s)}, \ldots, \theta_{p-1}^{(s)}, \mathbf{y})$.

↪ Then for s sufficiently large $(\theta_1^{(s)}, \ldots, \theta_p^{(s)}) \overset{\text{approx.}}{\sim} p(\theta_1, \ldots, \theta_p \mid \mathbf{y})$.

↪ The convergence of the p-tuple obtained at iteration s, $(\theta_1^{(s)}, \ldots, \theta_p^{(s)})$ to a draw from a joint posterior distribution occurs under mild regular conditions that are generally satisfied for most statistical models (see, e.g., Geman and Geman, 1984, or Roberts and Smith, 1993).

Bayesian computing

MCMC: Gibbs sampler

- Suppose we have data $y_1, \ldots, y_n$ such that $Y_i \overset{\text{iid}}{\sim} \text{N}(\mu, \sigma^2)$, where both μ and σ^2 are unknown.

- We need to specify the joint prior distribution $p(\mu, \sigma^2)$. Common choices are:

  1. $p(\mu, \sigma^2) = p(\mu \mid \sigma^2)p(\sigma^2)$.
  2. $p(\mu, \sigma^2) = p(\mu)p(\sigma^2)$.

- We will use the independent prior (second option) and assume $\mu \sim \text{N}(\mu_0, \sigma_0^2)$, and $\sigma^2 \sim \text{IG}(a, b)$.

- We have that

$$\mu \mid \sigma^2, \mathbf{y} \sim \text{N}\left(\frac{\frac{\mu_0}{\sigma_0^2} + \frac{n\bar{\mathbf{y}}}{\sigma^2}}{\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}}, \frac{1}{\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}} \right),$$

$$\sigma^2 \mid \mu, \mathbf{y} \sim \text{IG}\left(a + \frac{n}{2}, b + \frac{1}{2}\sum_{i=1}^{n}(y_i - \mu)^2 \right).$$

Bayesian computing

MCMC: Gibbs sampler

↪ The algorithm is then

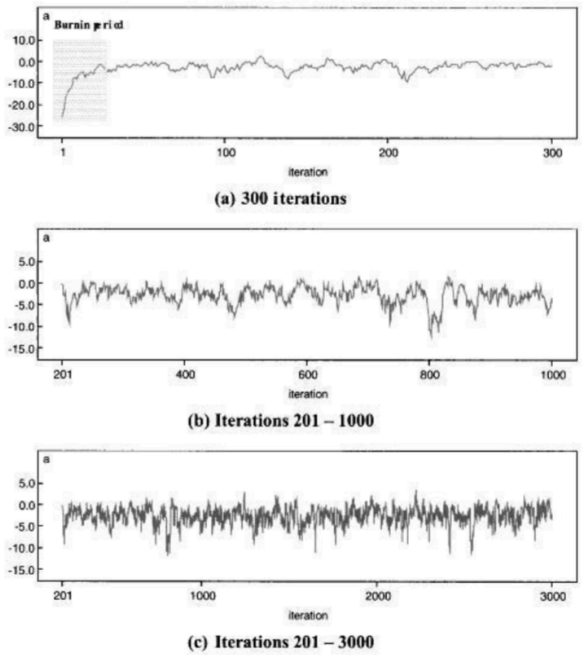
1. Set initial values $\mu^{(0)}$ and $(\sigma^{(0)})^2$.
2. For $s = 1, \ldots, S$
   - Draw $\mu^{(s)} \mid \mathbf{y}, (\sigma^{(s-1)})^2 \sim \text{N}\left(\frac{\frac{\mu_0}{\sigma_0^2} + \frac{n\bar{\mathbf{y}}}{(\sigma^{(s-1)})^2}}{\frac{1}{\sigma_0^2} + \frac{n}{(\sigma^{(s-1)})^2}}, \frac{1}{\frac{1}{\sigma_0^2} + \frac{n}{(\sigma^{(s-1)})^2}} \right)$.
   - Draw $(\sigma^{(s)})^2 \mid \mathbf{y}, \mu^{(s)} \sim \text{IG}\left(a + \frac{n}{2}, b + \frac{1}{2}\sum_{i=1}^{n}(y_i - \mu^{(s)})^2 \right)$.

↪ See implementation on the `R` script `example_gibbs_normal`.

Bayesian computing

MCMC: Gibbs sampler

- We will later in the lecture look more closely on convergence diagnostics.
- One basic diagnostic is to monitor the *traceplots*: the plots of the iteration versus the generated values of the parameters.
- If all values are within a zone without strong periodicity and (especially) tendencies, then there is no evidence of lack of convergence



Traceplots for (a) 300 iterations, (b) 1000 iterations after discarding the first 200, and (c) 3000 iterations after discarding the first 200. Image from Ntzoufras (2009).

Bayesian computing

MCMC: Gibbs sampler

- In the first trace plot, we can clearly see the burnin period (within the gray box), which must be discarded from the final sample. After this period the generated sampled values, are stabilised within a zone.

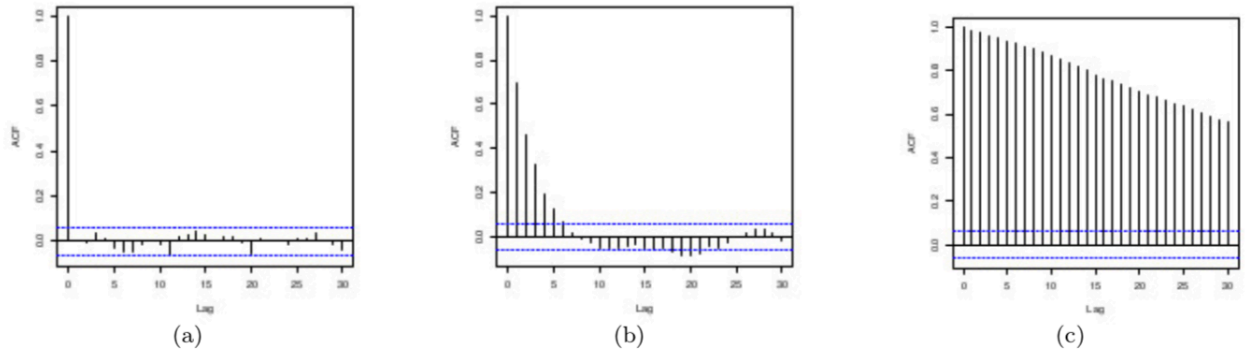
- In the second plot, the initial 200 iterations have been discarded to monitor the sampled values which demonstrate much better behaviour with small periodicities (up and down periods in the graph).

- Finally, generated values of the last trace plot are more convincing in terms of convergence, with all generated values within a parallel zone and no obvious tendencies or periodicities.

Bayesian computing

MCMC: Gibbs sampler

- ↪ Updating parameters one at a time can lead to high autocorrelation.
- ↪ The lag h autocorrelation function (ACF) for parameter θ_j is $\rho_j(h) = \text{Cor}(\theta_j^{(s)}, \theta_j^{(s-h)})$, i.e., it is the correlation between the given parameter value in the Markov chain separated by h iterations. The term $h > 1$ is usually referred to as lag.



Sample ACF plots representing (a) ideal mixing, (b) typical good mixing, (c) poor mixing. Image from prof Ruth King notes

Bayesian computing

MCMC: Gibbs sampler

- Note that the autocorrelation function is always equal to 1 for the value $h = 0$, since $\text{Cor}(\theta_j^{(s)}, \theta_j^{(s)}) = 1$.
- Ideally, for efficient Markov chains, there should be a fast decrease in the value of the autocorrelation function as the lag increases.
- This would imply that there is little relationship between values of the Markov chain within a small number of iterations.
- Conversely, poorly mixing chains will typically have a very shallow gradient in the ACF plot, with high autocorrelation values for even relatively large values of h.

Bayesian computing

MCMC: Gibbs sampler

- ↪ A somewhat crude, yet reasonably effective, method dealing with autocorrelation is to only keep every k draws from the posterior and discard the rest; this is known as thinning the chain.
- ↪ The advantages of thinning are both simplicity and a reduction in memory usage–saving and working with large chains can be burdensome.
- ↪ The disadvantage is that we are clearly throwing away information; thinning can never be as efficient as using all the iterations.

Bayesian computing

MCMC: Metropolis–Hastings algorithm

- ↪ Gibbs sampling requires drawing a sample from each full conditional distribution.
- ↪ In cases where the conditional distributions are conjugate sampling is straight-forward. But what if they are not conjugate?
- ↪ We could make draws from the conditional distributions using rejection sampling. This works well if there are only a few non-conjugate parameters but can be difficult to tune.
- ↪ Other methods have been proposed, with Metropolis–Hastings being the most widely used.

Bayesian computing

MCMC: Metropolis–Hastings algorithm

- ↪ Metropolis et al. (1953) originally formulated the Metropolis algorithm, and the paper was originally published in the physical sciences.
- ↪ Later, Hastings (1970) generalised the original method in what is known as the Metropolis–Hastings algorithm.
- ↪ The latter is considered to be the general formulation of all MCMC methods.
- ↪ Green (1995) further generalised the Metropolis–Hastings algorithm by introducing reversible jump Metropolis–Hastings algorithms for sampling from parameter spaces with different dimensions.

Bayesian computing

MCMC: Metropolis–Hastings algorithm

↪ The algorithm is summarised as follows:

1. Set initial values $\theta^{(0)}$.
2. For $s = 1, \ldots, S$, repeat the following steps:
   - Set $\boldsymbol{\theta} = \boldsymbol{\theta}^{(s-1)}$.
   - Draw candidate parameter values $\boldsymbol{\theta}^*$ from a proposal distribution $q(\boldsymbol{\theta}^* \mid \boldsymbol{\theta})$.
   - Calculate

$$\begin{aligned}\alpha(\boldsymbol{\theta}, \boldsymbol{\theta}^*) &= \min\left\{1, \frac{p(\boldsymbol{\theta}^* \mid \mathbf{y})q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^*)}{p(\boldsymbol{\theta} \mid \mathbf{y})q(\boldsymbol{\theta}^* \mid \boldsymbol{\theta})}\right\}, \\ &= \min\left\{1, \frac{f(\mathbf{y} \mid \boldsymbol{\theta}^*)p(\boldsymbol{\theta}^*)q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^*)}{f(\mathbf{y} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})q(\boldsymbol{\theta}^* \mid \boldsymbol{\theta})}\right\}.\end{aligned}$$

   - Generate $u \sim \text{Unif}(0, 1)$.
   - Set

$$\boldsymbol{\theta}^{(s)} = \begin{cases} \boldsymbol{\theta}^* & \text{if } u \leq \alpha, \\ \boldsymbol{\theta} & \text{if } u > \alpha. \end{cases}$$

Bayesian computing

MCMC: Metropolis–Hastings algorithm

- A special case of is the random walk Metropolis.
- In the original Metropolis algorithm, only symmetric proposal of the type $q(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}) = q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^*)$ were considered.
- Random walk Metropolis is a special case with $q(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}) = f(|\boldsymbol{\theta}^* - \boldsymbol{\theta}|)$.
- With such proposal distribution the kernel driving the chain is a random walk since the candidate values are of the form

$$\boldsymbol{\theta}^* = \boldsymbol{\theta} + \mathbf{z}, \quad \mathbf{z} \sim f.$$

- Both cases result in an acceptance probability that depends only on the posterior (target) distribution

$$\alpha(\boldsymbol{\theta}, \boldsymbol{\theta}^*) = \min\left\{1, \frac{f(\mathbf{y} \mid \boldsymbol{\theta}^*)p(\boldsymbol{\theta}^*)}{f(\mathbf{y} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})}\right\}.$$

- A usual proposal of this type is a multivariate normal $q(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}) = \mathrm{N}_p(\boldsymbol{\theta}, \mathbf{S}_{\boldsymbol{\theta}})$.

Bayesian computing

MCMC: Metropolis–Hastings algorithm

- As an implementation example, we consider the case where $y_1, \ldots, y_n$ comes from a normal distribution with known variance and unknown mean. The prior for the mean is normally distributed. Of course, as we have already seen, we do not need Metropolis type of algorithms to sample from the posterior in this case, but it serves as an illustrative example.

- Implementation available in the `R` script `example_random_walk_metropolis`.

Bayesian computing

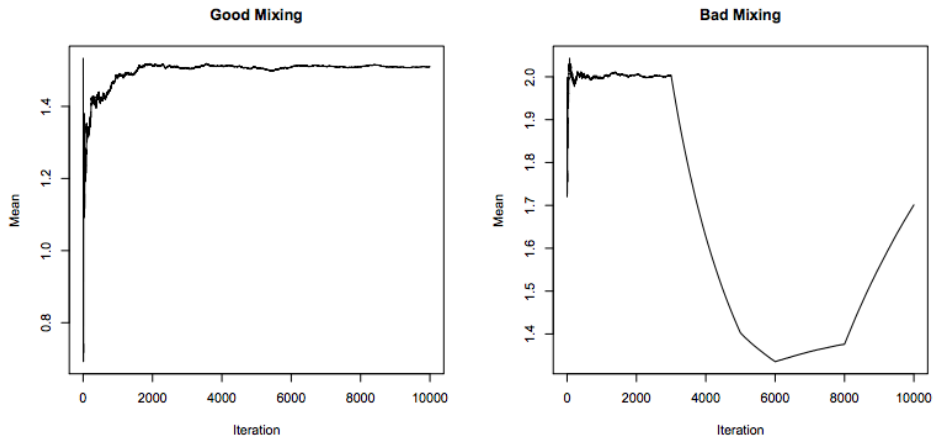
More on MCMC convergence diagnostics

- From the theory of Markov chains (more details on Bayesian theory course notes), we expect the chains to eventually converge to the stationary distribution, which is also our target distribution. Here the target is the posterior distribution.
- However, there is no guarantee that the chain has converged after S draws.
- How do we know whether the chain has actually converged?
- We can never be sure, but there are several tests we can do to check if the chain appears to be converged.

Bayesian computing

More on MCMC convergence diagnostics

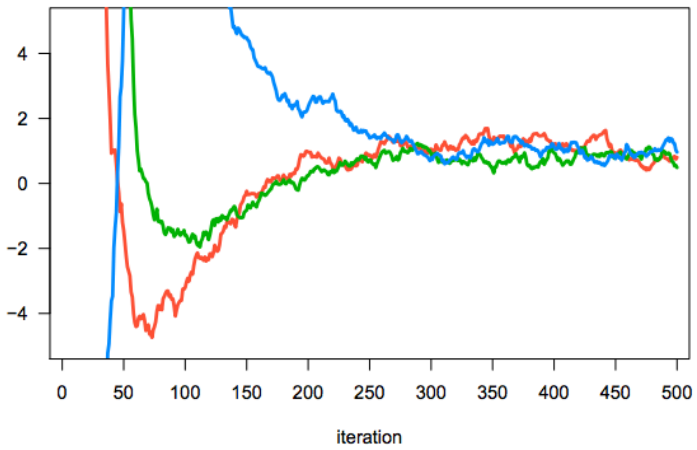
- We've already seen the use of traceplots and autocorrelation functions.
- Another quick check is to look at the running mean plots to check how well our chains are mixing.
- A running mean plot is a plot of the iterations against the mean of the draws up to each iteration.



Bayesian computing

More on MCMC convergence diagnostics

↪ Most approaches for detecting convergence, both formal and informal, rest on the idea of starting multiple Markov chains and observing whether they come together and start to behave similarly (if they do, we can pool the results from each chain).



↪ This plot indicates that the three chains converge to the posterior after around $S = 300$ iterations; certainly, however, the number of iterations required to reach convergence depends on the initial values.

↪ It is typically recommended (e.g., Gelman and Rubin, 1992) to use overdispersed initial values, meaning 'more variable than the target distribution' i.e., the posterior.

Bayesian computing

More on MCMC convergence diagnostics

- Although looking at trace plots is certainly useful, it is also desirable to obtain an objective, quantifiable measure of convergence.

- Numerous methods exist, although we will focus on the measure originally proposed in Gelman and Rubin (1992).

- The basic idea is to quantify the between-chain and the within-chain variability of a quantity of interest. If the chains have converged, these measures will be similar; otherwise, the between-chain variability will be larger.

Bayesian computing

More on MCMC convergence diagnostics

↪ The basic idea of the estimator is as follows (the actual estimator makes a number of modifications to account for degrees of freedom):

- Let B denotes the standard deviation of the pooled sample of all MS iterations, where M stands for the number of chains and S for the number of iterations on each chain.
- Let W denotes the average of the within-chain standard deviations.
- Quantify convergence with

$$\widehat{R} = \frac{B}{W}.$$

↪ If $\widehat{R} \gg 1$, this is clear evidence that the chains have not converged.

↪ As $S \to \infty$, $\widehat{R} \to 1$; $\widehat{R} < 1.05$ is widely accepted as implying convergence for practical purposes.

Bayesian computing

More on MCMC convergence diagnostics

↪ More details (along with other convergence measures) are given in the `coda` package, whose `gelman.diag` function provides, in addition to $\widehat{R}$ itself:

- An upper confidence interval for $\widehat{R}$.
- A multivariate extension of $\widehat{R}$ for quantifying convergence of the entire posterior.

Bayesian computing

More on MCMC convergence diagnostics

- ↪ The obvious downside to running multiple chains is that it is inefficient: we intentionally force our sampler to spend extra time in a non-converged state, which in turn requires much more burn-in.
- ↪ The obvious upside, however, is that it provides us with some measure of confidence that we are actually drawing samples from the posterior.
- ↪ But we reiterate that (without additional assumptions about the posterior) no method can truly prove convergence; diagnostics can only detect failure to converge.

Bayesian computing

More on MCMC convergence diagnostics

- We may use $\widehat{R}$, then, as a guide to how long we must run our chains until convergence.
- The obvious next question is: how long must we run our chains to obtain reasonably accurate estimates of the posterior?
- If we could obtain iid draws from the posterior, estimating the Monte Carlo standard error (at least, of the posterior mean) is straightforward: letting σ_{post} denote the posterior standard deviation, the MCSE is $\sigma_{\text{post}}/\sqrt{S}$.
- But, this will underestimate the true MC standard error due to autocorrelation in the samples generated using MCMC.
- Various remedies to obtain better estimate of MC error, with possibly the most popular being the *batches* mean method.