# Revisiting the Gelman-Rubin Diagnostic

Christina Knudson

April 3, 2019

## 1 Introduction

The Gelman-Rubin (GR) diagnostic has been one of the most popular diagnostics for MCMC convergence. The GR diagnostic framework relies on $m$ parallel chains ($m \geq 1$), each run for $n$ steps. The GR statistic (denoted $\hat{R}$) is the square root of the ratio of two estimators for the target variance. In finite samples, the numerator overestimates this variance and the denominator underestimates it. Each estimator converges to the target variance, meaning that $\hat{R}$ converges to 1 as $n$ increases. When $\hat{R}$ becomes sufficiently close to 1, the GR diagnostic declares convergence.

## 2 Effective Sample Size

For an estimator, Effective Sample Size (ESS) is the number of independent samples with the same standard error as a correlated sample.

The following is an expression for the lugsail-based psrf $\hat{R}_L^p$ for $m$ chains, each of length $n$ with p components.

$$\hat{R}_L^p = \sqrt{\left(\frac{n-1}{n}\right) + \frac{m}{\widehat{\text{ESS}}_L}},$$

Rearranging this yields an estimator of effective sample size:

$$\widehat{\mathrm{ESS}}_L = \frac{m}{\left(\hat{R}_L^p\right)^2 - \left(\frac{n-1}{n}\right)}.$$

*Remark* 1. **Doots!!** I remember we discussed and mentioned that the total ESS is not just the sum of the individual chains' ESSs. But this kind of is. So... am I using the wrong equation?

*Remark* 2. **?** explain that a minimum simulation effort must be set to safeguard from premature termination due to early bad estimates of $\sigma^2$. We concur and suggest a minimum simulation effort of $n = M_{\alpha,\epsilon,p}$.

**To Do**

- ■ Add ESS calculation (now in `gr.diag` and `n.eff`).
- ■ Add to `n.eff`.
  1. Call `target.psrf` using the info in their `mcmc.list` using defaults.
  2. Compare ESS to output of target.psrf and tell them whether this is sufficient for convergence.
  3. If insufficient, calculate how many more samples needed using

$$\frac{n_{\text{current}}}{n_{\text{current eff}}} \approx \frac{n_{\text{target}}}{n_{\text{target eff}}} \quad \Longrightarrow \quad \frac{n_{\text{current}}\ n_{\text{target eff}}}{n_{\text{current eff}}} \approx n_{\text{target}}.$$

  4. Added args of target.psrf.
- ☐ Create two functions: one to calculate $\hat{S}$ and another for $\hat{T}_L$. These two functions will be in the package but not exported.
- ☐ Rewrite `n.eff` to call these functions (rather than calling `gr.diag`).
- ☐ Add an example.

# 3 Package building

**To do:**

- ☐ Clean up documentation
- ☐ Add an example to `gr.diag`
- ☐ Check citations
- ☐ Check descriptions and theory
- ☐ Read through manual (pdf version)

# 4 Moving away from coda

Due to known coding errors (e.g. the miscalculated confidence interval for $\hat{R}$) and issues with `coda`, we would like to change our package (functions including `n.eff` and `gr.diag`) to no longer rely on `coda`. An incomplete list of ways we rely on `coda`:

- ☐ Users are required to input an `mcmc.list` in our current version. We should replace this with a list such that each object in the list is a matrix representing a single Markov chain: each row is one iteration and each column is one variable)
- ☐ An `mcmclist` has `mcmc` objects. We will replace each mcmc object with a matrix (as described in the previous bullet). We need to keep in mind that `mcmc` performs several input checks, so we will need to perform our own checks. For example, we will need to check that all objects are of class matrix, and that the dims of each object in the list are identical. (do.call might be useful for this?)
- ☐ Our code calls `niter` to find `n`; we can replace this with the number of rows in the first object of the list. I choose the first since all the objects should have the same dims so it wouldn't matter which we choose. We know we will have at least one Markov chain so we will have a first.
- ☐ We call `nvar` to find `p`; we can replace this with the number of columns in the

first object of the list. See note in previous bullet.

- ☐ `varnames` is currently pointlessly called
- ☐ `nchain` currently is used to determine the number of chains. This is easy enough to replace: just use `length` of the list.