Revisiting the Gelman-Rubin Diagnostic

Christina Knudson

September 8, 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 Package building

To do:

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

3 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.

3.1 `n.eff` version 1.0

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:

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

Remark 1. Vats et al (2018) explain that a minimum simulation effort must be set to safeguard from premature termination due to early bad estimates of σ^2. We concur and suggest a minimum simulation effort of $n = M_{\alpha,\epsilon,p}$.

Completed work

- Added ESS calculation (now in `gr.diag` and `n.eff`).
- Added to `n.eff`.
    1. Called `target.psrf` using the info in their `mcmc.list` using defaults.
    2. Compared 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.

The way version 1.0 works: first ESS is calculated using the univariate psrf. Then, if the user says `multivariate = TRUE` and if `Nvar>1` (meaning it really is a multivariate chain), then we enter an `if` statement where a bunch of stuff is calculated (including a new mpsrf-based ESS, which then replaces the univariate psrf-based ESS).

3.2 `n.eff` version 2.0

Version 2.0 of `n.eff` is all about NOT calling `gr.diag`, as that function performs calculations irrelevent to `n.eff`. Replacing version 1.0 with version 2.0 requires a bit of care: we want to use the univariate info in the univariate setting and the multivariate info in the multivariate setting.

Relevant univariate equation:

Relevant multivariate equation:

$$mn\left(\frac{\det(\hat{S})}{\det(\hat{T}_L)}\right)^{1/p}$$

To create version 2.0 of `n.eff`, the following work must be done:

- ■ asymptotic variances
  - write function ✓
  - documentation ✓
  - test ✓(tests stayed same before/after implementation)

- asymptotic variance matrix
  - write function ✓
  - documentation ✓
  - test ✓(tests stayed same before/after implementation)
- sample variances s and variance matrix
  - write function ✓
  - test ✓(tests stayed same before/after implementation)
- Rewrite `stable.GR` to call these functions
- Rewrite `n.eff` to call these functions (rather than calling `stable.gr`).
- Add an example.

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.
- 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.

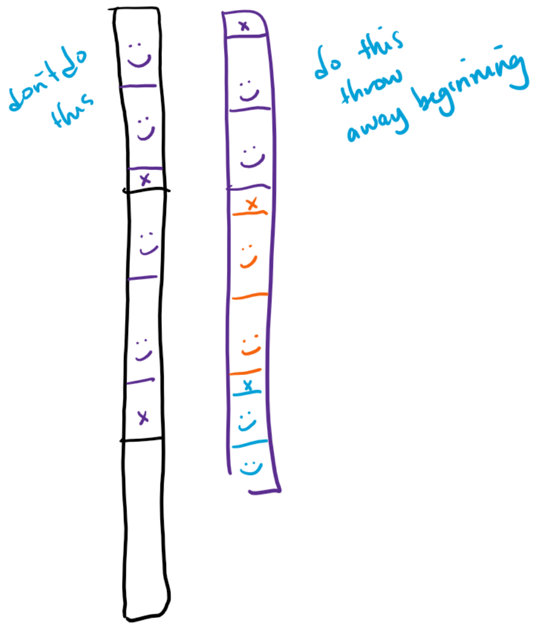
- We call `nvar` to find `p`; we can replace this with the number of columns in the first object of the list. See note in previous bullet.
- `varnames` is currently pointlessly called
- `nchain` currently is used to determine the number of chains. This is easy enough to replace: just use `length` of the list.

5 Implementing replicated batch means

Consider the parameter $\tau_n^2 = nVar(\bar{X}_{i\cdot})$. In our first pass at estimator τ_n^2, we used the average of each chain's batch means variance estimates. Now, we are moving towards using replicated batch means: essentially, we will pool the chains together and use the floor of the square root of a *single* chain's length (rather than the conglomerated Monte Carlo sample size.) With the `mcmcse` function `mcse.mat`, it's easy to specify the batch size with the `size` argument.

We have to think a little more because `mcse.mat` (and `mcse.multi`) automate some things that we will now have to consider. In particular, `mcse.mat` (and `mcse.multi`) takes care of choosing which Monte Carlo samples to trim off in order to use the correct batch size. Now we will need to:

1. Calculate the correct Monte Carlo batch size (floor of square root of a single chain's length).
2. Split each chain into batches by trimming the samples at the start of the chain rather than at the tail of the chain (see sketch below).
3. Stack the chains on top of each other (rbind).
4. Pass the conglomerated chain to mcse.mat (and also pass the correct batch size).



These calculations should be done in `asym.var`, since this function calls `mcse.mat` (and `mcse.multi`). We also need to implement replicated batch means for matrix T (the multivariate version of τ_n^2), which calls `mcse.multi` rather than `mcse.mat`.

To do list:

- ☐ Implement replicated batch means for the univariate PSRF
- ☐ Implemented replicated batch means for the MPSRF