The NIMBLE system

NIMBLE is:

- A system for using algorithms on hierarchical statistical models (defined by BUGS code).
- A system for programming algorithms to be used on hierarchical models.
- A partial compiler for math in R, and A bridge to BUGS and JAGS systems.
- Why use NIMBLE rather than other packages?
- Customize your MCMC; choose samplers and block arbitrarily.
- Define your own distributions and functions for use in BUGS code.
- Apply algorithms other than MCMC to a BUGS-defined model: SMC, particle filter, MCMC, etc.
- Write your own algorithms (including MCMC samplers) that can be used on any BUGS model.
- Disaggregate algorithms via an R package containing specific algorithm code and a dependency on the NIMBLE package.

Outline of the poster

Here we’ll focus on using the system to improve MCMC performance, considering three strategies:

- Customizing your MCMC with user-chosen samplers and blocking (left column)
- Automating blocking of parameters (center column)
- Writing your own MCMC sampler, which can then be used on any model (right column)

Using NIMBLE for tuning an MCMC

Example model: litters

There are 2 calls to a group of rat litters, with X = 10 litters (i.e., mothers), in each group, and a variable number of pups in each litter. Survival in a group is in a litter is governed by a survival probability for each litter, rj, with an exchangeable beta prior within each group.

The model shows poor MCMC performance because of (1) dependence between hyperparameters for a given group and (2) dependence between hyperparameters and associated random effects of a given group.

Define a default MCMC as in JAGS and BUGS

Set up the model (based on the BUGS code) and a default MCMC. This involves a few more steps than in JAGS or BUGS, but with the benefit that NIMBLE gives you greater control over how you use the model and set up an MCMC (or other algorithms).

The automated blocking procedure is available within NIMBLE by calling

sampler_RW_reflect <- nimbleFunction(
  contains = sampler_BASE,
  ... )

Here are trace plots for the default MCMC, showing poor mixing and strong dependence within pairs of hyperparameters.

Choose the blocking scheme

NIMBLE allows users to swap out samplers and choose blocking schemes. Here we’ll block the hyperparameters for each group.

Use a specialized sampler

Blocking worked better but didn’t account for dependence of the hyperparameters with their associated random effects (cross-level dependence). We’ll replace the basic block sampler with a more sophisticated sampler that samples hyperparameters and associated random effects jointly, sampling

Next steps

- Explore strategy to justify retaining samples during adaptation phase.
- Automate exploration of sampler choices (including reparameterization using fast blocks).
- In exploration phase, update multiple updates on a single parameter or multiple blocks within an iteration to enable more efficient optimization amongst the updates.

Automating MCMC sampler choices

Motivation

- Adaptive random walk block Metropolis has been successful in providing tuned proposals that account for dependence.
- But the choices of what samplers to use and what parameters to block are still done manually by the analyst.
- Goal is to provide algorithms that search space of valid MCMC samplers for better MCMC performance.
- Key metric is effective sample size per computing time.

Trading off MCMC and computational efficiency

For a dimensionless parameter, there are advantages and drawbacks of choosing between the use of autotuning samplers and a single dimensionless block update.

MCMC performance results for suite of example models

Table: Effective sample size (ESS) is measured in effective samples per 10,000 iterations. Runtime is presented as seconds per 10,000 iterations, and Efficiency is effective sample size produced per second of runtime.

Automating MCMC

Automating MCMC processing flow

Writing an algorithm: a simple example

How to write a basic objective function (which could be passed to an optimization function such as R’s optim.

- setup: code needs which nodes are dependent, which nodes are independent.
- run code: compute log probability density for all relevant nodes.