compareMCMCs

an R package for running, managing, and comparing results from different MCMC packages

Sally Paganin

June 6, 2023

Acknowledgements

Perry de Valpine (UC Berkeley)

Daniel Turek (Williams College)

Introduction

Markov chain Monte Carlo (MCMC) are methods used to simulate from complicated probability distributions

- Widely used in Bayesian statistical analysis
  ➡️ target distribution: posterior distribution of parameters given data.
- There are many MCMC algorithms (a.k.a. samplers)
  - Metropolis-Hastings (conjugate samplers)
  - Hamiltonian Monte Carlo (HMC)
  - Multivariate samplers
  - ... new methods are constantly developed
- Methods are also implemented differently in different software

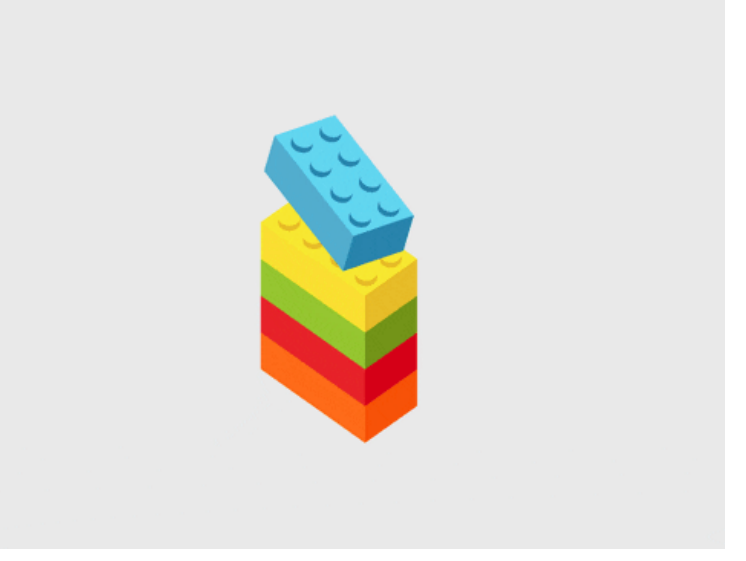


Plenty of possibilities...

- Samplers can be combined

  i.e. use a different algorithm for each parameter

- Implementations of the same algorithm can differ between software packages



Questions

1. Which algorithms are "better" for a given model?
2. Which software implements better an algorithm?

Efficiency in MCMC

Algorithmic mixing & computational speed

- Mixing measured in terms of **Effective Sample Size** (ESS) equivalent number of independent samples
    - R packages implementing different estimators for the ESS
      `coda`, `mcmcse`, `batchmeans`, `stableGR` ...
- Speed
    - sampling time vs total time (e.g. MCMC vs HMC)
    - language of implementation (e.g. R vs C++)

Efficiency = ESS/(time in seconds)

compareMCMCs

Package for managing performance comparisons among MCMC software packages

Features

- the `compareMCMCs` function: run one or more MCMCs and manage the results.
- the `MCMCresult` class to manage results
- a plugin system to include new MCMC engines;
- a plugin system for new metrics for comparison among MCMCs;
- a system for applying parameter conversions, in case difference MCMCs use different parameterizations and/or parameter names;
- a system for generating html pages with figures from comparison metrics, including a plugin system to provide new page components;

Supported software

- Probabilistic programming languages
  ➡️ programming paradigm for automated inference

  - `jags` (`rjags`) - uses mostly on slice samplers
  - `stan` (`rstan`) - Hamiltonian Monte Carlo
  - `nimble` - suite of different algorithms

A toy example

- adaptive M-H with nimble (default)
- slice sampling with nimble

- slice sampling with JAGS (default)
- HMC with Stan (default)

A toy example

```
library(compareMCMCs)
library(nimble)

# This model code will be used for both nimble
# and JAGS
modelCode <- nimbleCode({
    a ~ dunif(0, 100)
    y ~ dgamma(a, 2)
})

modelInfo <- list(code = modelCode, constants = list(y = 2),
    inits = list(a = 1))

# Here is a custom MCMC configuration function
# for nimble
configure_nimble_slice <- function(model) {
    configureMCMC(model, onlySlice = TRUE)
}

res_nimble <- compareMCMCs(modelInfo, MCMCs = c("nimble",
    "nimble_slice"), nimbleMCMCdefs = list(nimble_slice = "configure_nir
    MCMCcontrol = list(inits = list(a = 1), niter = 4000,
        burnin = 200))
```

Define the same model in Stan

```
stan_code <- c("data {real y;}",
               "parameters {real a;}",
               "model {target += uniform_lpdf(a | 0, 100);",
               "       target += gamma_lpdf(y | a, 2);}")

## make two lists to provide stan model and arguments for sampling
stan_model_args <- list(model_code = stan_code)

stan_sampling_args <- list(data = list(y=2),
                           init = list(list(a=1)), # one chain
                           warmup = 200,
                           iter = 2000)
```

Run all models in series

```
res <- compareMCMCs(modelInfo,
                    MCMCs = c('jags',
                              'nimble',       # nimble default
                              'nimble_slice', # nimble slice
                              'stan'),
                    nimbleMCMCdefs =
                      list(nimble_slice = 'configure_nimble_slice'),
                    MCMCcontrol = list(inits = list(a = 1),
                                       niter = 2000,
                                       burnin = 200),
                    externalMCMCinfo = list(stan = list(
                      stan_model_args = stan_model_args,
                      sampling_args = stan_sampling_args)))
```

or combine them later

```
res <- c(res_jags, res_nimble, res_nimble_slice, res_stan)
```

`MCMCresult`

Results are stored in an `R6` class (encapsulate OOP & self-modifiable)

- `MCMC`: optional name for the MCMC method.
- `samples`: matrix of MCMC samples (iteration X parameters)
- `times`: a list of times including elements for setup, burn-in, postburn-in (sampling for recorded samples), and sampling (normally burn-in + postburn-in).
- `metrics`: a list of MCMC performance metrics (ESS, efficency, parameter summaries). Organized `byMCMC`, `byParameter`, `other` (not used)

`make_MCMC_comparison_pages()`

Function to create an html output with comparisons of MCMC results - here an example

Comparison metrics

```
res$nimble$metrics
> $byMCMC
>     MCMC min_efficiency mean_efficiency
> 1 nimble       255278.2        255278.2
> $byParameter
>     MCMC Parameter     mean   median       sd CI95_low CI95_upp
> 1 nimble         a 5.056558 4.931279 2.057118 1.552509 9.306015
>   ESS     efficiency
> 1 765.8347  255278.2
> $other
> list()
```

Add new metrics

Function that takes as input a `MCMCresult` object and outputs a list

```
MCMCmetric_median <- function(result, ...) {
    res <- apply(result$samples, 2, median)
    list(byParameter = list(median = res))
}

addMetrics(res, list(MCMCmetric_median))
```

Page plug-in: figure or text component

A page plug-in is a list with up to five elements:

- `make`: name of a function to create the plottable output such as a ggplot object.
- `fileSuffix`: is a character suffix for figures name
- `linkText`: hyperlink at the top of the comparison page.
- `plot`: name of a function that plot the output to a jpeg file. The function takes as input the `plottable` element of the list returned from `make`.
- `control`: is a list that will be passed to the make function.

Registering the component

```
registerPageComponents(

  list(myNewComponent =
         list(make = "myMakeFunction",
              fileSuffix = "_myPageComponent",
              linkText = "My new page component.")
       )
  )
```

More about `make`

`make` names a function with 2 arguments

1. tidy metrics from `combineMetrics(res, include_times = TRUE)`
2. `control` element of the plugin (user defined)

Return information

1. figure component
   The plotting is done between via call to `jpeg` and `dev.off()`
   - `plottable` - an object that can be plot
   - `height`
   - `width`
2. text component
   - `printable`- character string of html or output from the `xtable` package (that can be plotted as html)

New MCMC plugins

An interface to a new MCMC engine is provided as a function that runs the algorithms and return and `MCMCresult` object

- `MCMCinfo`: The element of `externalMCMCinfo` named to match this MCMC plugin. This can contain whatever information is needed for the plugin.
- `MCMCcontrol`: The `MCMCcontrol` argument to `compareMCMCs`.
- `monitorInfo`: A list of names of parameters to monitor (record) in MCMC output.
- `modelInfo` : The `modelInfo` argument to `compareMCMCs`. If the call to `compareMCMCs` involved creating a `nimble` model, it will be added to this list with the name `model`.

Status

- version of the package on CRAN & Github
  https://cran.r-project.org/web/packages/compareMCMCs/index.html
  https://github.com/nimble-dev/compareMCMCs/tree/master
- Paper on Journal of Open Source Software
  https://joss.theoj.org/papers/10.21105/joss.03844
- Online vignette

Development

- Improve documentation
- Add usage examples
- Any suggestion?