

nimbleHMC: An R package for Hamiltonian Monte Carlo sampling in nimble

Daniel Turek¹, Perry de Valpine², and Christopher J. Paciorek²

¹ Lafayette College, USA ² University of California, USA ¶ Corresponding author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: 

Submitted: 14 January 2024

Published: unpublished

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/))

Summary

Markov chain Monte Carlo (MCMC) algorithms are widely used for fitting hierarchical models to data. MCMC is the predominant tool used in Bayesian analyses to generate samples from the posterior distribution of model parameters conditional on observed data. MCMC is not a single algorithm, but rather a framework in which various sampling methods (samplers) are assigned to operate on subsets of unobserved parameters. There exists a vast set of valid samplers to draw upon, which differ in complexity, autocorrelation of samples produced, and applicability.

Hamiltonian Monte Carlo [HMC; Radford M. Neal (2011)] sampling is one such technique, applicable to continuous-valued parameters, which uses gradients to generate large transitions in parameter space. The resulting samples have low autocorrelation, and therefore have high information content, relative for example to an equal-length sequence of highly autocorrelated samples. The No-U-Turn (NUTS) variety of HMC sampling [HMC-NUTS; Hoffman & Gelman (2014)] greatly increases the usability of HMC by introducing a recursive tree of numerical integration steps that makes it unnecessary to pre-specify a fixed number of steps. Hoffman & Gelman (2014) also introduce a self-tuning scheme for the step size, resulting in a fully automated HMC sampler with no need for manual tuning.

Many software packages offer implementations of MCMC, such as nimble (de Valpine et al., 2017), WinBUGS (Lunn et al., 2000), jags (Plummer, 2003), pyMC (Fonnesbeck et al., 2015), NumPyro (Phan et al., 2019), TensorFlow Probability (Pang et al., 2020), and Stan (Carpenter et al., 2017), among others. These packages differ, however, in their approaches to sampler assignments. As sampling techniques vary in computation and quality of the samples, the effectiveness of the MCMC algorithms will vary depending on the software and model.

Among MCMC software packages, nimble is distinct in allowing easy customization of sampler assignments from a high-level interface. Users may assign any valid samplers to each parameter or group of parameters, selecting from samplers provided with nimble or samplers they have written in nimble's algorithm programming system. Samplers provided with nimble include random walk Metropolis-Hastings sampling (Robert & Casella, 1999), slice sampling (Radford M. Neal, 2003), elliptical slice sampling (Murray et al., 2010), automated factor slice sampling (Tibbitts et al., 2014), conjugate sampling (George et al., 1993), and others.

The nimbleHMC package provides implementations of two versions of HMC-NUTS sampling for use within nimble, both written in nimble's algorithm programming system within R. Specifically, nimbleHMC provides the original ("classic") HMC-NUTS algorithm as developed in Hoffman & Gelman (2014), and a modern version of HMC-NUTS sampling matching the HMC sampler available in version 2.32.2 of Stan (Stan Development Team, 2023). The samplers provided in nimbleHMC can be assigned to any continuous-valued parameters, and may be used in combination with other samplers provided with nimble.

Example

The following example demonstrates fitting a hierarchical model to data using `nimbleHMC`. We use the European Dipper (*Cinclus cinclus*) dataset drawn from ecological capture-recapture (e.g., [Lebreton et al., 1992](#); [Turek et al., 2016](#)). Modelling includes both continuous parameters to undergo HMC sampling and discrete parameters that cannot be sampled via HMC. We are not aware of other software that supports this combination other than `nimbleHMC`.

Individual birds are captured, tagged, and potentially recaptured on subsequent sighting occasions. Data is a 294×7 binary-valued array of capture histories of 294 uniquely tagged birds over 7 years. Model parameters are detection probability (p), and annual survival rates on non-flood years (ϕ_1) and flood years (ϕ_2). Data is provided in the R package `mra` ([McDonald, 2018](#)).

```
library(mra)
data(dipper.data)
y <- dipper.data[,1:7]
```

We specify the hierarchical model using uniform priors on the interval $[0, 1]$ for all parameters. Binary-valued latent states $x_{i,t}$ represent the true alive (1) or dead (0) state of individual i on year t . Doing so allows the survival process to be modelled as $x_{i,t+1} \sim \text{Bernoulli}(\phi_{f_t} \cdot x_{i,t})$ where f_t indicates the flood/non-flood history of year t , and observations are modelled as $y_{i,t} \sim \text{Bernoulli}(p \cdot x_{i,t})$.

```
library(nimbleHMC)

code <- nimbleCode({
  phi[1] ~ dunif(0, 1)
  phi[2] ~ dunif(0, 1)
  p ~ dunif(0, 1)
  for(i in 1:N) {
    for(t in (first[i]+1):T) {
      x[i,t] ~ dbern(phi[f[t]] * x[i,t-1])
      y[i,t] ~ dbern(p * x[i,t])
    }
  }
})
```

A `nimble` model object is now built. The argument `buildDerivs = TRUE` results in under-the-hood support for obtaining derivatives from model calculations, as necessary for derivative-based HMC sampling.

```
Rmodel <- nimbleModel(
  code,
  constants = list(N = nrow(y), T = ncol(y), f = c(1,2,2,1,1,1,1),
    first = apply(y, 1, which.max)),
  data = list(y = y),
  inits = list(phi = c(0.5, 0.5), p = 0.5, x = array(1, dim(y))),
  buildDerivs = TRUE)
```

Next we create an MCMC configuration object, which specifies the sampling algorithm to be applied to each parameter. By default, `configureMCMC` uses `nimble`'s default sampler assignments of adaptive random walk Metropolis-Hastings [RW sampler; Robert & Casella (1999)] for each parameter, and a binary Gibbs sampler for each $x_{i,t}$ latent state.

```
conf <- configureMCMC(Rmodel)

## RW sampler (3)
## - phi[] (2 elements)
```

```

92 ## - p
93 ## binary sampler (848)
94 ## - x[] (848 elements)

95 Now we customize the MCMC configuration object to use HMC sampling for the model
96 parameters. replaceSamplers replaces the samplers operating on  $\phi_1$ ,  $\phi_2$  and  $p$  with the
97 modern HMC-NUTS sampler (called the NUTS sampler) provided in nimbleHMC. The classic
98 version of the HMC-NUTS sampler could be assigned by specifying type = "NUTS_classic".

99 conf$replaceSamplers(target = c("phi", "p"), type = "NUTS")
100 conf$printSamplers(byType = TRUE)
101
102 ## NUTS sampler (1)
103 ## - phi, p
104 ## binary sampler (848)
105 ## - x[] (848 elements)

106 Alternatively, the convenience function configureHMC could be used to create an identical
107 MCMC configuration, applying HMC-NUTS sampling to  $\phi_1$ ,  $\phi_2$  and  $p$ , and default binary
108 samplers for discrete parameters.

109 Now we build and compile the MCMC algorithm.

110 Rmcmc <- buildMCMC(conf)
111 Cmodel <- compileNimble(Rmodel)
112 Cmcmc <- compileNimble(Rmcmc, project = Rmodel)

113 We execute the MCMC for 20,000 iterations, and discard the initial 10,000 samples as burn-in.

114 set.seed(0)
115 samples <- runMCMC(Cmcmc, niter = 20000, nburnin = 10000)

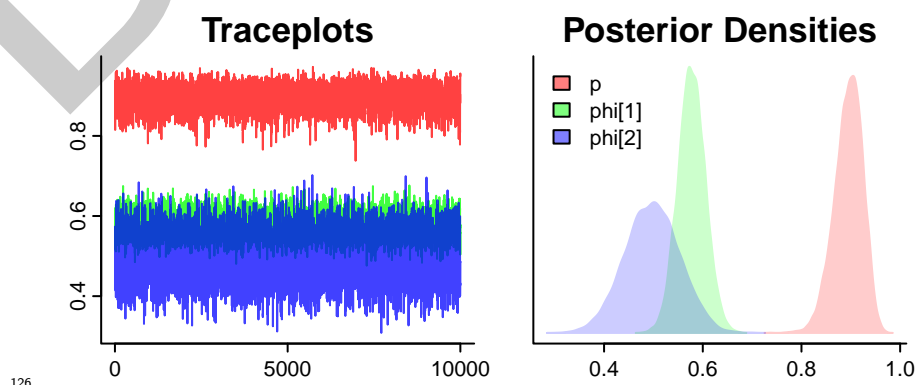
116 Finally, posterior summary statistics are calculated for the model parameters.

117 samplesSummary(samples, round = 2)
118
119 ##           Mean Median St.Dev. 95%CI_low 95%CI_upp
120 ## p           0.89   0.90   0.03     0.83     0.94
121 ## phi[1]       0.58   0.58   0.03     0.52     0.63
122 ## phi[2]       0.50   0.50   0.06     0.39     0.60

123 Traceplots and posterior density plots are generated using the samplesPlot function from the
124 basicMCMCplots package.

125 basicMCMCplots::samplesPlot(samples, legend.location = "topleft")

```



Statement of need

HMC is recognized as a state-of-the-art MCMC sampling algorithm. As testimony to this, software packages such as Stan exclusively employ HMC sampling. Consequently, such software cannot operate on models containing discrete parameters (upon which HMC cannot operate). Models with discrete parameters arise in a range of statistical motifs including hidden Markov models, finite mixture models, and generally in the presence of unobserved categorical data (Bartolucci et al., 2022). In contrast, other mainstream MCMC packages (WinBUGS, OpenBUGS and jags) can sample discrete parameters, but provide no facilities for HMC sampling. This leaves a gap, as there is no support for HMC sampling of hierarchical models that also contain discrete parameters.

nimbleHMC fills this gap, by providing two HMC samplers that operate inside nimble's MCMC engine. The base nimble package provides a variety of MCMC sampling algorithms, as well as the ability to customize MCMC sampler assignments. nimbleHMC augments the set of sampling algorithms provided in nimble with two options for HMC sampling, which can be used alongside any other samplers. The example presented here demonstrates precisely that: HMC sampling operating alongside discrete samplers, which is not possible without the use of nimbleHMC.

Which combination of samplers will optimize MCMC efficiency for any particular problem is an open question. One metric of comparison is the effective sample size of the samples generated per unit runtime of the algorithm, which quantifies how quickly an MCMC algorithm generates information about parameter posteriors. This metric is studied in Turek et al. (2017) and Ponisio et al. (2020), with the conclusion that the best sampling strategy is problem-specific rather than universal. For that reason, the ability to mix-and-match samplers from a large pool of candidates is important from both practical and theoretical standpoints. Indeed, packages such as compareMCMCs (de Valpine et al., 2022) exist specifically to compare the relative performance of MCMC algorithms. The addition of HMC sampling provided by nimbleHMC supports new practical combinations for applied MCMC, as well as facilitates a deeper study of Bayesian modelling.

References

- Bartolucci, F., Pandolfi, S., & Pennoni, F. (2022). Discrete latent variable models. *Annual Review of Statistics and Its Application*, 9, 425–452.
- Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., & Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of Statistical Software*, 76(1). <https://doi.org/10.18637/jss.v076.i01>
- de Valpine, P., Paganin, S., & Turek, D. (2022). compareMCMCs: An r package for studying MCMC efficiency. *Journal of Open Source Software*, 7(69), 3844. <https://doi.org/10.21105/joss.03844>
- de Valpine, P., Turek, D., Paciorek, C. J., Anderson-Bergman, C., Lang, D. T., & Bodik, R. (2017). Programming with models: Writing statistical algorithms for general model structures with NIMBLE. *Journal of Computational and Graphical Statistics*, 26(2), 403–413. <https://doi.org/10.1080/10618600.2016.1172487>
- Fonnesbeck, C., Patil, A., Huard, D., & Salvatier, J. (2015). PyMC: Bayesian stochastic modelling in python. *Astrophysics Source Code Library*, ascl–1506.
- George, E. I., Makov, U., & Smith, A. (1993). Conjugate likelihood distributions. *Scandinavian Journal of Statistics*, 147–156.
- Hoffman, M. D., & Gelman, A. (2014). The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1), 1593–1623.

- 174 Lebreton, J.-D., Burnham, K. P., Clobert, J., & Anderson, D. R. (1992). Modeling survival
175 and testing biological hypotheses using marked animals: A unified approach with case
176 studies. *Ecological Monographs*, 62(1), 67–118. <https://doi.org/10.2307/2937171>
- 177 Lunn, D. J., Thomas, A., Best, N., & Spiegelhalter, D. (2000). WinBUGS—a bayesian modelling
178 framework: Concepts, structure, and extensibility. *Statistics and Computing*, 10, 325–337.
- 179 McDonald, T. (2018). *Mra: Mark-recapture analysis*. [https://CRAN.R-project.org/package=](https://CRAN.R-project.org/package=mra)
180 [mra](https://CRAN.R-project.org/package=mra)
- 181 Murray, I., Adams, R., & MacKay, D. (2010). Elliptical slice sampling. *Proceedings of the*
182 *Thirteenth International Conference on Artificial Intelligence and Statistics*, 541–548.
- 183 Neal, Radford M. (2003). Slice sampling. *The Annals of Statistics*, 31(3), 705–767. <https://doi.org/10.1214/aos/1056562461>
184 <https://doi.org/10.1214/aos/1056562461>
- 185 Neal, Radford M. (2011). *Handbook of markov chain monte carlo* (pp. 113–162). Chapman;
186 Hall/CRC.
- 187 Pang, B., Nijkamp, E., & Wu, Y. N. (2020). Deep learning with tensorflow: A review. *Journal*
188 *of Educational and Behavioral Statistics*, 45(2), 227–248.
- 189 Phan, D., Pradhan, N., & Jankowiak, M. (2019). Composable effects for flexible and accelerated
190 probabilistic programming in NumPyro. *arXiv Preprint arXiv:1912.11554*.
- 191 Plummer, M. (2003). JAGS: A program for analysis of bayesian graphical models using
192 gibbs sampling. *Proceedings of the 3rd International Workshop on Distributed Statistical*
193 *Computing*, 124, 1–10.
- 194 Ponisio, L. C., de Valpine, P., Michaud, N., & Turek, D. (2020). One size does not fit
195 all: Customizing MCMC methods for hierarchical models using NIMBLE. *Ecology and*
196 *Evolution*, 10(5), 2385–2416. <https://doi.org/10.1002/ece3.6053>
- 197 Robert, C. P., & Casella, G. (1999). The metropolis—hastings algorithm. In *Monte carlo*
198 *statistical methods* (pp. 231–283). Springer.
- 199 Stan Development Team. (2023). *Stan modeling language users guide and reference manual,*
200 *version 2.32.2*. <https://mc-stan.org>
- 201 Tibbits, M. M., Groendyke, C., Haran, M., & Liechty, J. C. (2014). Automated factor slice
202 sampling. *Journal of Computational and Graphical Statistics*, 23(2), 543–563. <https://doi.org/10.1080/10618600.2013.791193>
203 <https://doi.org/10.1080/10618600.2013.791193>
- 204 Turek, D., de Valpine, P., & Paciorek, C. J. (2016). Efficient markov chain monte carlo
205 sampling for hierarchical hidden markov models. *Environmental and Ecological Statistics*,
206 23, 549–564. <https://doi.org/10.1007/s10651-016-0353-z>
- 207 Turek, D., de Valpine, P., Paciorek, C. J., & Anderson-Bergman, C. (2017). Automated
208 parameter blocking for efficient markov chain monte carlo sampling. *Bayesian Analysis*,
209 12(2), 465–490. <https://doi.org/10.1214/16-BA1008>