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

Daniel Turek [1¶], **Perry de Valpine** [2], **and Christopher J. Paciorek**[2]

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

DOI: 10.21105/joss.06745

Software

- Review
- Repository
- Archive

Editor: Johanna Bayer

Reviewers:

- @matt-graham
- @larryshamalama

Submitted: 14 January 2024
Published: 10 July 2024

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.

A key design feature of `nimble`'s MCMC is to allow 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 (Tibbits 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.

Individual birds are captured, tagged, and potentially recaptured on subsequent sighting occasions. Data is a 255×7 binary-valued array of capture histories of 255 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), and individuals which are first sighted on the final (7^{th}) sighting occasion do not contribute to inference, and are removed from the sighting histories.

```
library(mra)
data(dipper.data)
dipper <- dipper.data[,1:7]
y <- dipper[apply(dipper, 1, which.max) < 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. The model structure conditions on the first observation of each individual, where first_i is the first observation period of individual i, and x_{i,first_i} is assigned the value one. 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) {
        x[i,first[i]] <- 1
        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)
##   - p
## binary sampler (848)
##   - x[]  (848 elements)
```

Now we customize the MCMC configuration object to use HMC sampling for the model parameters. `replaceSamplers` replaces the samplers operating on ϕ_1, ϕ_2 and p with the modern HMC-NUTS sampler (called the `NUTS` sampler) provided in `nimbleHMC`. The classic version of the HMC-NUTS sampler could be assigned by specifying `type = "NUTS_classic"`.

```
conf$replaceSamplers(target = c("phi", "p"), type = "NUTS")
conf$printSamplers(byType = TRUE)
```

```
## NUTS sampler (1)
##   - phi, p
## binary sampler (848)
##   - x[]  (848 elements)
```

Alternatively, the convenience function `configureHMC` could be used to create an identical MCMC configuration, applying HMC-NUTS sampling to ϕ_1, ϕ_2 and p, and default binary samplers for discrete parameters.

Now we build and compile the MCMC algorithm.

```
Rmcmc <- buildMCMC(conf)
Cmodel <- compileNimble(Rmodel)
Cmcmc <- compileNimble(Rmcmc, project = Rmodel)
```

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

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

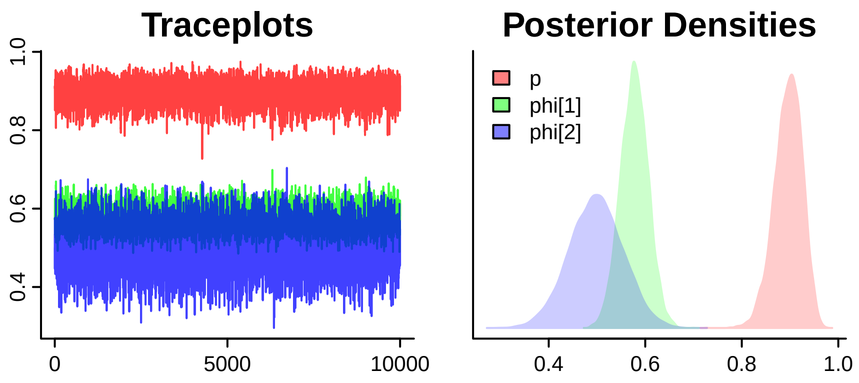
Finally, posterior summary statistics are calculated for the model parameters.

```
samplesSummary(samples, round = 2)
```

```
##        Mean Median St.Dev. 95%CI_low 95%CI_upp
## p      0.90   0.90    0.03      0.84      0.95
## phi[1] 0.58   0.58    0.03      0.52      0.63
## phi[2] 0.50   0.50    0.06      0.39      0.61
```

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

```
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), or would require marginalization of the likelihood to remove these discrete dimensions from the sampling problem. 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 the use case of HMC sampling of hierarchical models that also contain discrete parameters.

`nimbleHMC` accomplishes this, 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. https://doi.org/10.1146/annurev-statistics-040220-091910

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.

Lebreton, J.-D., Burnham, K. P., Clobert, J., & Anderson, D. R. (1992). Modeling survival and testing biological hypotheses using marked animals: A unified approach with case studies. *Ecological Monographs*, *62*(1), 67–118. https://doi.org/10.2307/2937171

Lunn, D. J., Thomas, A., Best, N., & Spiegelhalter, D. (2000). WinBUGS-a bayesian modelling framework: Concepts, structure, and extensibility. *Statistics and Computing*, *10*, 325–337. https://doi.org/10.1023/A:1008929526011

McDonald, T. (2018). *Mra: Mark-recapture analysis*. https://CRAN.R-project.org/package=mra

Murray, I., Adams, R., & MacKay, D. (2010). Elliptical slice sampling. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 541–548.

Neal, Radford M. (2003). Slice sampling. *The Annals of Statistics*, *31*(3), 705–767. https://doi.org/10.1214/aos/1056562461

Neal, Radford M. (2011). *Handbook of markov chain monte carlo* (pp. 113–162). Chapman; Hall/CRC.

Pang, B., Nijkamp, E., & Wu, Y. N. (2020). Deep learning with tensorflow: A review. *Journal of Educational and Behavioral Statistics*, *45*(2), 227–248. https://doi.org/10.3102/1076998619872761

Phan, D., Pradhan, N., & Jankowiak, M. (2019). Composable effects for flexible and accelerated probabilistic programming in NumPyro. *arXiv Preprint arXiv:1912.11554*. https://doi.org/10.48550/arXiv.1912.11554

Plummer, M. (2003). JAGS: A program for analysis of bayesian graphical models using gibbs sampling. *Proceedings of the 3rd International Workshop on Distributed Statistical Computing*, *124*, 1–10.

Ponisio, L. C., de Valpine, P., Michaud, N., & Turek, D. (2020). One size does not fit all: Customizing MCMC methods for hierarchical models using NIMBLE. *Ecology and Evolution*, *10*(5), 2385–2416. https://doi.org/10.1002/ece3.6053

Robert, C. P., & Casella, G. (1999). The metropolis—hastings algorithm. In *Monte carlo statistical methods* (pp. 231–283). Springer.

Stan Development Team. (2023). *Stan modeling language users guide and reference manual, version 2.32.2*. https://mc-stan.org

Tibbits, M. M., Groendyke, C., Haran, M., & Liechty, J. C. (2014). Automated factor slice sampling. *Journal of Computational and Graphical Statistics*, *23*(2), 543–563. https://doi.org/10.1080/10618600.2013.791193

Turek, D., de Valpine, P., & Paciorek, C. J. (2016). Efficient markov chain monte carlo sampling for hierarchical hidden markov models. *Environmental and Ecological Statistics*, *23*, 549–564. https://doi.org/10.1007/s10651-016-0353-z

Turek, D., de Valpine, P., Paciorek, C. J., & Anderson-Bergman, C. (2017). Automated parameter blocking for efficient markov chain monte carlo sampling. *Bayesian Analysis*, *12*(2), 465–490. https://doi.org/10.1214/16-BA1008