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

Daniel Turek, Perry de Valpine, and Christopher J. Paciorek

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

DOI: 10.21105/joss.06745

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., 2019), 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 $255 \times 7$ 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 ($\phi_1$) and flood years ($\phi_2$). 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.

```r
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 $x_{i,\text{first}_i}$ is assigned the value one. Observations are modelled as $y_{i,t} \sim \text{Bernoulli}(p \cdot x_{i,t})$.

```r
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.

```r
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.

```r
conf <- configureMCMC(Rmodel)
```

Now we customize the MCMC configuration object to use HMC sampling for the model parameters. `replaceSamplers` replaces the samplers operating on \( \phi_1 \), \( \phi_2 \) 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"`.

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

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

Now we build and compile the MCMC algorithm.

```r
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.

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

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

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

```r
## 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.

```r
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.

\texttt{nimbleHMC} accomplishes this, by providing two HMC samplers that operate inside \texttt{nimble}'s MCMC engine. The base \texttt{nimble} package provides a variety of MCMC sampling algorithms, as well as the ability to customize MCMC sampler assignments. \texttt{nimbleHMC} augments the set of sampling algorithms provided in \texttt{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 \texttt{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 \texttt{compareMCMCs} (de Valpine et al., 2022) exist specifically to compare the relative performance of MCMC algorithms. The addition of HMC sampling provided by \texttt{nimbleHMC} supports new practical combinations for applied MCMC, as well as facilitates a deeper study of Bayesian modelling.

References


