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

23 March 2023

Summary

Markov chain Monte Carlo (MCMC) algorithms are widely used for fitting hierarchical (graphical) models to observed data, and more generally, for simulating from high-dimensional probability distributions. MCMC is the predominant tool used in Bayesian analyses, where the distribution of interest (the "target distribution") is defined as the posterior distribution of the unknown model parameters conditional on the data. MCMC does not specify a single algorithm, but rather a family of algorithms admitting any assignment of valid sampling techniques ("samplers") to the unobserved model dimensions. There exist a vast and diverse landscape of valid samplers to draw upon, which differ significantly in their underlying approaches to the sampling problem, complexity, autocorrelation of the samples produced, and applicability.

Hamiltonian Monte Carlo (HMC; Brooks et al. 2011) is one such sampling technique which can be applied to any subset of continuous-valued model dimensions. HMC uses the gradient of the target distribution to generate large transitions (in parameter space) in the output sequence of samples. This results in low autocorrelation, and therefore high information content. That is, the samples generated using HMC are more likely to be highly informative about the target distribution of interest, relative for example to an equal-length sequence of highly autocorrelated samples. This rich information content does not come freely, however, as calculating gradients of the target distribution is computationally expensive.

There exist numerous software packages which provide implementations of MCMC for mainstream use such as `nimble` (de Valpine et al. 2017), `WinBUGS` (Lunn et al. 2000), `jags` (Plummer 2003), `pyMC` (Fonnesbeck et al. 2015), and `Stan` (Carpenter et al. 2017). Each such package provides a language for specifying general hierarchical model structures, and supplying data. Given a hierarchical model and associated data, each package generates an MCMC algorithm customized to generate samples from the posterior distribution of the specified model, which is then executed to generate a large number of samples. These packages differ, however, in their approaches to sampler assignment for each unobserved model dimension. As sampling techniques vary in terms of computational demands and the quality of the samples produced, the effectiveness of the MCMC algorithms may vary depending on the software used, and the particular model at hand. Each software package provides a valid, but distinct approach for assigning samplers to define an MCMC algorithm.

Among general-purpose MCMC software packages, `nimble` uniquely provides the ability to specify which samplers are applied to each model dimension. Prior to generating an executable MCMC algorithm, `nimble` has the intermediate stage of MCMC configuration. At configuration time, users may select any valid assignment of samplers to each unobserved model dimension, selecting among the suite of samplers provided with `nimble`. The base `nimble` package provides a variety of non-derivative-based sampling options, including random walk Metropolis-Hastings sampling (Robert and Casella 1999), slice sampling (Neal 2003), elliptical slice sampling (Murray, Adams, and MacKay 2010), automated factor slice sampling (Tibbits et al. 2014), conjugate sampling (George, Makov, and Smith 1993), and many others. After sampler configuration is complete, an MCMC algorithm is generated according to the specified sampler assignments, and executed to generate a sequence of posterior samples.

The `nimbleHMC` package provides an implementation of HMC sampling which is compatible for use within `nimble`. Specifically, `nimbleHMC` implements the No-U-Turn variety of HMC (HMC-NUTS; Hoffman and Gelman 2014), which removes the necessity of hand-specifying tuning parameters of the HMC sampler. Using `nimbleHMC`, HMC samplers can be assigned to any subset of continuous-valued model dimensions at the time

of nimble's MCMC configuration, which may be used in combination with any other samplers provided with the base nimble package.

Example

The following provides an example R session of fitting a hierarchical model to data via HMC sampling using nimbleHMC. The specific dataset is the well-studied European Dipper *Cinclus cinclus)* dataset drawn from ecological capture-recapture (*e.g.*, Lebreton et al. 1992; Turek, Valpine, and Paciorek 2016). This example is selected to involve both continuous-valued parameters to undergo HMC sampling and discrete latent states which cannot be sampled via HMC. This combination is not supported by software other than nimbleHMC.

Here, individual birds are captured, tagged, and potentially recaptured on subsequent annual sighting occasions. Data is a 294×7 binary-valued array of capture histories of 294 uniquely tagged birds over 7 years, where an example row (sighting history) of {0,0,1,1,0,1,0} represents an individual that was first sighted and tagged in year 3, and subsequently resighted on years 4 and 6. 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). First we load the dataset, where columns 1-7 contain the 294 sighting histories.

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

Next, we specify the hierarchical model to define relationships between parameters and data. nimble adopts and extends the hierarchical modelling language from WinBUGS and jags. Flat uniform priors on the interval $[0, 1]$ are assigned for all parameters. We use binary-valued latent states $x_{i,t}$ to 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])
        }
    }
})
```

Using the model specification, a nimble model object is now built using the nimbleModel function. In addition to the hierarchical specification, we also provide a list of constant values (number of individuals, number of years, flood year indicator, and year of first capture for each individual), a specification of the data, and a list of initial values for model parameters and latent states. The latent $x_{i,t}$ values are all initialized as one, which is guaranteed to be a valid initial state. In addition, the argument buildDerivs = TRUE affects derivatives of likelihood calculations to be built into the model object, to support derivative-based algorithms operating on the model – here, 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 MCMC sampling algorithms to be applied to each unobserved model dimension. The function `configureMCMC` creates an MCMC configuration object using `nimble`'s default sampler assignments. This assigns an adaptive random walk Metropolis-Hastings sampler (`RW` sampler; Robert and Casella 1999) to each model parameter, and a Gibbs sampler customized for binary-valued distributions (`binary` sampler) for each $x_{i,t}$ latent state.

Output from `configureMCMC` indicates three instances of the `RW` sampler, and 848 instances of the `binary` sampler. These correspond to the 848 observation occasions which succeed the initial observation of each individual.

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

```
## RW sampler (3)
##   - phi[]  (2 elements)
##   - p
## binary sampler (848)
##   - x[]  (848 elements)
```

Now we customize the MCMC configuration object to instead use HMC sampling for the model parameters. We use the `replaceSamplers` method to replace current samplers operating on p, ϕ_1 and ϕ_2 instead with the `HMC` sampler provided in the `nimbleHMC` package. The `printSamplers` method is used to display the the modified sampler assignments.

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

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

Now we build an executable MCMC algorithm using `buildMCMC`, and compile the model object and MCMC algorithm to C++ for fast execuction.

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

The compiled MCMC algorithm is executed using `runMCMC`. We execute the MCMC for a single chains of 20,000 iterations, and discard the initial 10,000 samples as burn-in.

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

```
##  [Note] HMC sampler (nodes: phi[1], phi[2], p) is using 1000 warmup iterations.
##  [Note] HMC sampler (nodes: phi[1], phi[2], p) encountered 3 divergent paths.
```

The HMC sampler outputs two notes, indicating the number of warmup iterations and the total number of divergent paths encountered (see Hoffman and Gelman 2014 for details).

Posterior summary statistics are calculated using the `samplesSummary`, which agree with those of Lebreton et al. (1992).

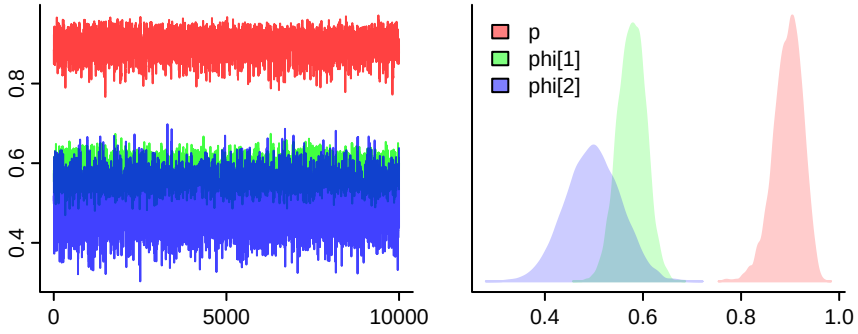
```
samplesSummary(samples, round = 2)
```

```
##        Mean Median St.Dev. 95%CI_low 95%CI_upp
```

```
## p      0.90   0.90    0.03      0.83      0.94
## phi[1] 0.58   0.58    0.03      0.52      0.63
## phi[2] 0.50   0.50    0.06      0.39      0.61
```

We also generate traceplots and posterior density plots using the `samplesSummary` 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 strategy, capable of efficiently generating samples with strong inferential power. A testimony to this, software packages such as `Stan` have been built exclusively around the use of HMC sampling. As a result, however, such software is unable to operate on models with discrete dimensions, a result of the non-applicability of HMC to non-continuous dimensions. Models with discrete-valued parameters arise in a range of common statistical motifs including hidden Markov models, finite mixture models, and generally in the presence of unobserved categorical data, among others (Bartolucci, Pandolfi, and Pennoni 2022). In contrast, other mainstream MCMC packages such as `WinBUGS`, `OpenBUGS` and `jags` have the ability to sample discrete dimensions, but provide no facilities for HMC sampling. This leaves a gap, as there is no support for applying HMC sampling to continuous-valued dimensions of hierarchical models which also contain discrete dimensions.

`nimbleHMC` fills this gap, by providing an HMC sampler which operates seamlessly inside `nimble`'s native MCMC engine. `nimble` provides a host of MCMC sampling algorithms which are suitable for either continuous-valued or discrete-valued dimensions, as well as the ability to customize an MCMC algorithm by specifying which algorithm(s) are assigned to sample each dimension. `nimbleHMC` supplements the suite of sampling algorithms provided with `nimble` with the addition of an HMC sampler, which can be used in concert with other samplers on other continuous or discrete-valued dimensions. The example presented herein demonstrates precisely this use case: HMC sampling operating alongside other samplers in a model containing discrete-valued dimensions, a task which is not possible without the use of `nimbleHMC`.

It is an open question what MCMC algorithm, or which combination of samplers, will optimize the fitting of any particular hierarchical model and dataset. One metric of comparison is the effective sample size of the sequence of samples (which quantifies the information content of the sample) generated per unit runtime of the algorithm. That is, how quickly an MCMC algorithm generates meaningful information to characterize the target distribution. This metric is studied in works such as Turek et al. (2017) and Ponisio et al. (2020), but what assignment of samplers maximizes this metric is an open question. For that reason, the ability to mix-and-match samplers from among as large a pool of candidates as possible is important from both practical and theoretical standpoints. Indeed, there even exist packages such as `compareMCMCs` (de Valpine, Paganin, and Turek 2022), the purpose of which is to compare the relative performance of distinct MCMC algorithms. The addition of HMC sampling provided by `nimbleHMC` supports new combinations of MCMC algorithms, as well as facilitates a deeper study of practical Bayesian modelling.

References

Bartolucci, Francesco, Silvia Pandolfi, and Fulvia Pennoni. 2022. "Discrete Latent Variable Models." *Annual Review of Statistics and Its Application* 9: 425–52.

Brooks, Steve, Andrew Gelman, Galin Jones, and Xiao-Li Meng. 2011. *Handbook of Markov Chain Monte Carlo*. CRC press.

Carpenter, Bob, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. "Stan: A Probabilistic Programming Language." *Journal of Statistical Software* 76 (1).

de Valpine, Perry, Sally Paganin, and Daniel Turek. 2022. "CompareMCMCs: An R Package for Studying Mcmc Efficiency." *Journal of Open Source Software* 7 (69): 3844.

de Valpine, Perry, Daniel Turek, Christopher J Paciorek, Clifford Anderson-Bergman, Duncan Temple Lang, and Rastislav Bodik. 2017. "Programming with Models: Writing Statistical Algorithms for General Model Structures with Nimble." *Journal of Computational and Graphical Statistics* 26 (2): 403–13.

Fonnesbeck, Chris, Anand Patil, David Huard, and John Salvatier. 2015. "PyMC: Bayesian Stochastic Modelling in Python." *Astrophysics Source Code Library*, ascl–1506.

George, Edward I, UE Makov, and AFM Smith. 1993. "Conjugate Likelihood Distributions." *Scandinavian Journal of Statistics*, 147–56.

Hoffman, Matthew D, and Andrew Gelman. 2014. "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo." *J. Mach. Learn. Res.* 15 (1): 1593–1623.

Lebreton, Jean-Dominique, Kenneth P Burnham, Jean Clobert, and David R Anderson. 1992. "Modeling Survival and Testing Biological Hypotheses Using Marked Animals: A Unified Approach with Case Studies." *Ecological Monographs* 62 (1): 67–118.

Lunn, David J, Andrew Thomas, Nicky Best, and David Spiegelhalter. 2000. "WinBUGS-a Bayesian Modelling Framework: Concepts, Structure, and Extensibility." *Statistics and Computing* 10: 325–37.

McDonald, Trent. 2018. *Mra: Mark-Recapture Analysis*. https://CRAN.R-project.org/package=mra.

Murray, Iain, Ryan Adams, and David MacKay. 2010. "Elliptical Slice Sampling." In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 541–48. JMLR Workshop; Conference Proceedings.

Neal, Radford M. 2003. "Slice Sampling." *The Annals of Statistics* 31 (3): 705–67.

Plummer, Martyn. 2003. "JAGS: A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling." In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing*, 124:1–10. 125.10. Vienna, Austria.

Ponisio, Lauren C, Perry de Valpine, Nicholas Michaud, and Daniel Turek. 2020. "One Size Does Not Fit All: Customizing Mcmc Methods for Hierarchical Models Using Nimble." *Ecology and Evolution* 10 (5): 2385–2416.

Robert, Christian P, and George Casella. 1999. "The Metropolis—Hastings Algorithm." In *Monte Carlo Statistical Methods*, 231–83. Springer.

Tibbits, Matthew M, Chris Groendyke, Murali Haran, and John C Liechty. 2014. "Automated Factor Slice Sampling." *Journal of Computational and Graphical Statistics* 23 (2): 543–63.

Turek, Daniel, Perry de Valpine, Christopher J Paciorek, and Clifford Anderson-Bergman. 2017. "Automated Parameter Blocking for Efficient Markov Chain Monte Carlo Sampling." *Bayesian Analysis* 12 (2): 465–90.

Turek, Daniel, Perry de Valpine, and Christopher J Paciorek. 2016. "Efficient Markov Chain Monte Carlo Sampling for Hierarchical Hidden Markov Models." *Environmental and Ecological Statistics* 23: 549–64.