Introduction to the R package NIMBLE: **N**umerical **I**nference of statistical **M**odels for **B**ayesian and **L**ikelihood **E**stimation

Blaine Mooers, Ph.D.

Department of Biochemistry & Molecular Biology

College of Medicine

OUHSC

Statistical Computing User Group

OUHSC

Dec. 6, 2016

Paper about NIMBLE

Perry de Valpine, Daniel Turek, Christopher J. Paciorek, Clifford Anderson-Bergman, Duncan Temple Lang, and Rastislav Bodik

(just accepted)

Programming with models: writing statistical algorithms for general model structures with NIMBLE

Journal of Computational and Graphical Statistics

arxiv.org/pdf/1505.05093.pdf

MCMC needed for high dimensional integrals

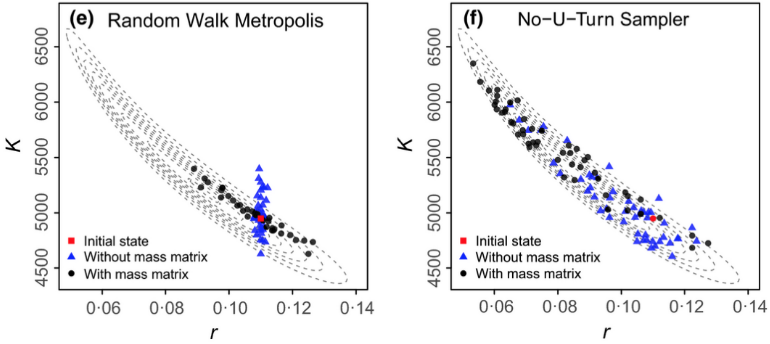
$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)} = \frac{L(\theta|y)p(\theta)}{p(y)}$$

$$p(\theta_i|y) = \frac{L(\theta_i|y)p(\theta_i)}{\int_k p(y|\theta_k)p(\theta_k)} \propto L(\theta_i|y)p(\theta_i)$$

$$p(\theta_1|y) = \int_{\theta_2} \cdots \int_{\theta_k} p(\theta|y) d\theta_2 \ldots d\theta_k$$

$$p(\tilde{y}|y) = \int p(\tilde{y}|\theta)p(\theta|y) d\theta$$

Monte Carlo Markov Chain



1

---

[1] Monnahan, C. C., Thorson, J. T., & Branch, T. A. (2016). Faster estimation of Bayesian models in ecology using Hamiltonian Monte Carlo. Methods in Ecology and Evolution. DOI: 10.1111/2041-210X.12681

Lack general software for hierarchical models

Topics hard to address with current software:

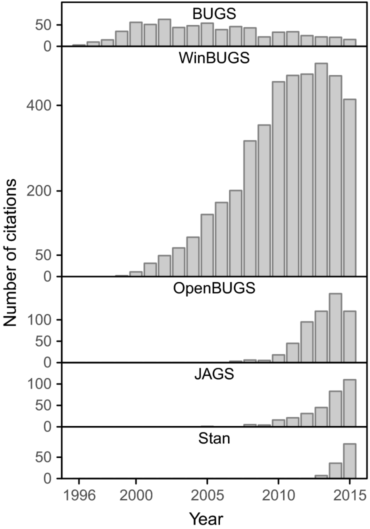
- improving performance of MCMC algorithms
- developing maximum likelihood methods
- new approximations of posterior distributions
- new methods of model assessment and selection
- new combinations of existing methods

NIMBLE's Goal: enable applying a variety of algorithms to any model defined as a directed acyclic graph (DAG).

Approaches to Bayesian Computing in Current Software

1. Provide a constrained family of models and algorithms customized to these models
2. Provide a language for model specification (BUGS family, Stan, Template Model Builder (TMB R-package), BayesX and PyMC).
3. NIMBLE: combines flexible model specification with functions that adapt to model structures.

Citations per year by software package



Source: Monnahan, C. C., Thorson, J. T., & Branch, T. A. (2016). Faster estimation of Bayesian models in ecology using Hamiltonian Monte Carlo. Methods in Ecology and Evolution. DOI: 10.1111/2041-210X.12681

MCMC software for Bayesian Analysis

1990 1995 2000 2005 2010 2015

BUGS WinBUGS **R2WinBUGS** **Nimble**

OpenBUGS **R2OpenBUGS**

STAN **RSTAN**

JAGS **RJAGS** **RStanArm**

Rethinking

BayesX **Shinystan**

MCMCpack **LaplacesDemon**

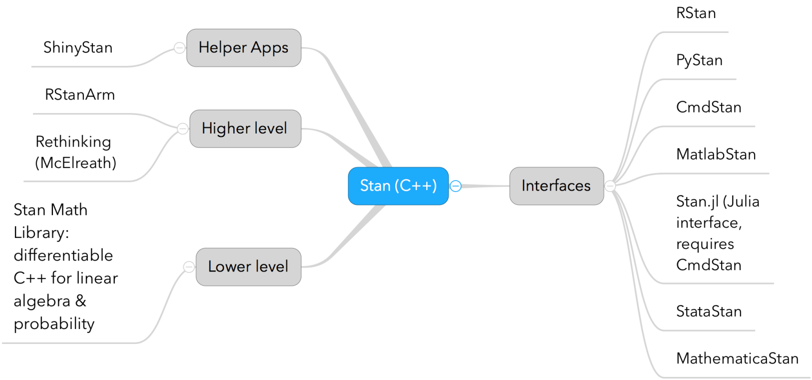
MultiNest

pyMC2 pyMC3

mamba.jl

JASP

Stan Ecosystem



NIMBLE's support for features of BUGS and JAGS

1. Stochastic and deterministic nodes
2. Most uni- and multivariate distributions in BUGS
3. Link functions
4. Most mathematical functions in BUGS
5. "for" loops for iterative declarations
6. Handles arrays of nodes in up to four dimensions
7. Truncation and censoring of MCMC as in JAGS.

Advancements in NIMBLE

Extensions to BUGS and JAGS:

1. User-defined nimbleFunctions() and distributions in the model code.
2. Alternate parameterizations (sigma or precision) for distributions
3. Can use named parameters for distributions and functions, similar to R function calls.

New features:

1. Processes BUGS code into a model object that can be quieried
2. Allows model generic programming by seperating setup steps from run-time steps
3. Includes a domain specific language (DSL) that is embedded within R

Possible ways to optimize performance

1. Several kinds of MCMC
2. Other Monte Carlo methods
   (e.g., Sequential Monte Carlo)
3. Different modular combinations of methods
   (e.g., particle filters and MCMC in state-space time-series models)
4. Algorithms for maximum likelihood estimation
5. Methods for models criticism and model selection
6. Estimation of prediction error
7. "Likelihood free" or "plug-and-play"
   synthetic likelihood, approximate Bayesian computation, or iterated filtering.
8. Parametric bootstrapping
9. Test same model and algorithm with multiple data sets in one script.

Models as programming objects

1. implementation of BUGS with extensions as a DSL
2. nimbleFunction() system for programming with models
3. NIMBLE compiler for model objects and nimbleFunctions()

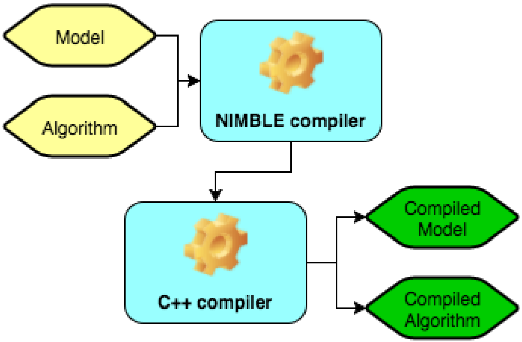


Figure: NIMBLE flow chart.

Source: https://bids.berkeley.edu/news/NIMBLE-programming-statistical-algorithms-graphical-hierarchical-models

Reasons to use or not to use NIMBLE

Five reasons to use NIMBLE

1. BUGS code –> model objects
2. User-customizd MCMC samplers
3. Can compile code in C^{++} without knowing C^{++}.
4. Output is returned to R.
5. User programmed methods possible, even methods without a BUGS model.

Four reasons not to use NIMBLE

1. JAGS may be faster for MCMCs that rely on Gibbs sampling
2. Hamiltonian MC in Stan may work better for some models
3. NIMBLE does not allow for stochastic indexing; use JAGS
4. NIMBLE takes a long time to build models with tens of thousands nodes
   (once built, the algorithm run times can be quite good).

Installation of NIMBLE

Need a C^{++} compiler already installed.
Depends on igraph.
`http://r-NIMBLE.org/`

install.packages(”NIMBLE”, repos = ”http://r-NIMBLE.org”, type = ”source”)

library(NIMBLE)

??NIMBLE

Documentation for NIMBLE

`http://r-NIMBLE.org/` 148 page manual

`https://github.com/NIMBLE-dev/NIMBLE-demos`

`http://www.mrc-bsu.cam.ac.uk/wp-content/uploads/WinBUGS_Vol1.pdf`

`http://www.mrc-bsu.cam.ac.uk/wp-content/uploads/WinBUGS_Vol2.pdf`

https://www.youtube.com/watch?v=I_yKe6WW76g YouTube video about NIMBLE

Bivariate example

```
library(nimble);library(igraph);library(ggplot2);library(ggExtra)
myBUGScode <- nimbleCode(mu   dnorm(0, sd = 100)
                 sigma   dunif(0, 100)
                 for(i in 1:10) y[i]   dnorm(mu, sd = sigma))
myModel <- nimbleModel(myBUGScode)
plot(myModel$getGraph())
myData <- rnorm(10, mean = 2, sd = 5)
myModel$setData(list(y = myData)) myModel$setInits(list(mu = 0, sigma = 1))
myMCMC <- buildMCMC(myModel)
compiled <- compileNimble(myModel, myMCMC)
compiled$myMCMC$run(10000)
samples <- as.matrix(compiled$myMCMC$mvSamples)
plot(density(samples[,’mu’]))
plot(density(samples[,’sigma’]))
plot(samples[ , ’mu’], type = ’l’,
                 xlab = ’Iteration’, ylab = expression(mu))
plot(samples[ , ’sigma’], type = ’l’,
                 xlab = ’Iteration’, ylab = expression(sigma))
```

Bivariate marginal posterior distributions

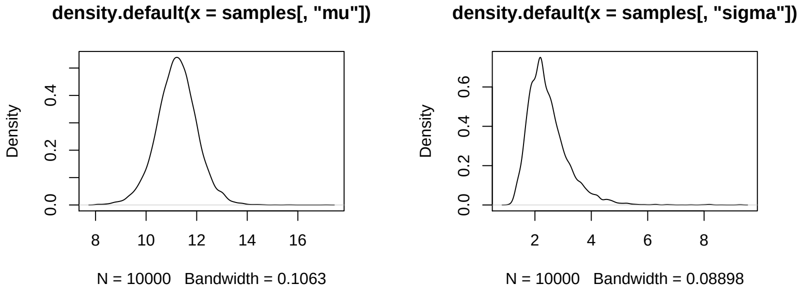
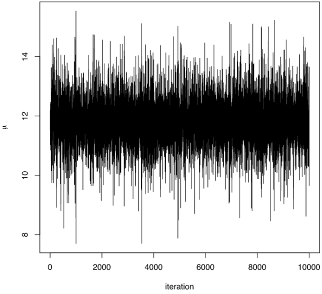


Figure: The posterior distributions of the mean and sigma.

Traceplots for mu and sigma



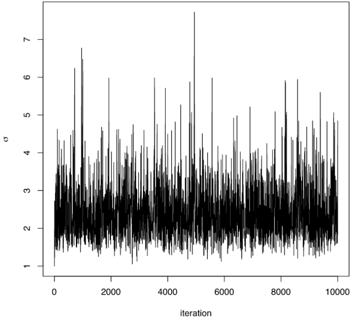
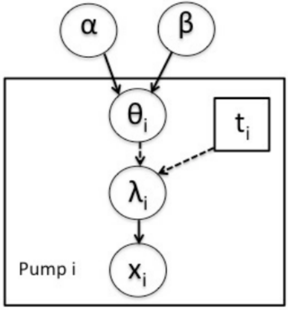


Figure: The posterior distributions of the mean and sigma.

BUGS pump model as directed acyclic graph



$N = 10$ pumps
x_i, the number of failures for pump i
t_i, the length of operation of pump i in thousands of hours
θ_i, the failure rate of pump i
x_i, assumed to follow a Poisson distribution with
mean number of failures $\lambda_i = \theta_i \times t_i$

Pump model in BUGS language

```
library(nimble)
pumpCode <- nimbleCode({
  for (i in 1:N){

      # likelihood (data model)
      x[i] ~ dpois(lambda[i])

      # latent process (random effects)
      # linear predictor
      lambda[i] <- theta[i]*t[i]
      # random effects distribution
      theta[i] ~ dgamma(alpha,beta)
  }
  # priors on hyperparameters
  alpha ~ dexp(1.0)
  beta ~ dgamma(0.1,1.0)
})
```

Pump model written in R code

```
N <- 10
t <- c(94.3, 15.7, 62.9, 126, 5.24, 31.4, 1.05, 1.05, 2.1, 10.5)
x <- c(5, 1, 5, 14, 3, 19, 1, 1, 4, 22)
pumpConsts <- list(t = t, N = 10)
pumpData <- list(x = x)
pumpInits <- list(alpha = 1, beta = 1,
         theta = rep(0.1, pumpConsts$N))
pump <- nimbleModel(pumpCode,
          data = pumpData, constants = pumpConsts, inits = pumpInits)
```

Each node has a nimbleFunction

Three kinds of model nodes.

- **top** have no stochastic parents
- **end** have no stochastic dependents
- **latent** have stochastic parents and dependents

Each node has a **nimbleFunction** with four run-time functions.

- **calculate** Calculates log probability mass or density function for a stochastic node. Executes the computation, stores the result as the value of the node, and returns 0 for deterministic nodes, .
- **calculateDiff** returns for stochastic nodes the difference between the new log probability and the old previously stored log probability
- **simulate** generates a draw from the distribution for a stochastic node. Identical to calculate for deterministic nodes.
- **getLogProb** returns current log probability value for a stochastic node and returns 0 for a deterministic node

Distributions in NIMBLE

Distribution	Canonical name	Alias
Binomial	dbin	dbinom
Chi-square	dchisq	dchisqr
Dirichlet	ddirch	ddirich
Multinomial	dmulti	dmultinom
Negative binomial	dnegbin	dnbinom
Weibull	dweib	dweibull
Wishart	dwish	dwishart

Sampler Algorithms provided with NIMBLE

1. binary (Gibbs) sampler
2. scalar Metropolis-Hastings random walk **RW** sampler
3. conjugate (Gibbs) samplers
4. multivariate Metropolis-Hastings **RW** block sampler
5. **slice** sampler
6. elliptical slice sampling: **ess** sampler
7. hierarchical **crossLevel** sampler
8. customized log likelihood evaluations using the **RW_llFunction** sampler
9. terminal node **posterior_predictive** sampler
10. particle MCMC sampler

Summary

- NIMBLE DSL extends the BUG Language
- expands choice of MCMC samplers
- allows compiling code in C++
- models with $> 10^4$ parameters are slow to compile but run fast
- lacks stochastic indexing (i.e., indices that are not constants, use JAGS)