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

MCMC needed for high dimensional integrals

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

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

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

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

MCMC software and R

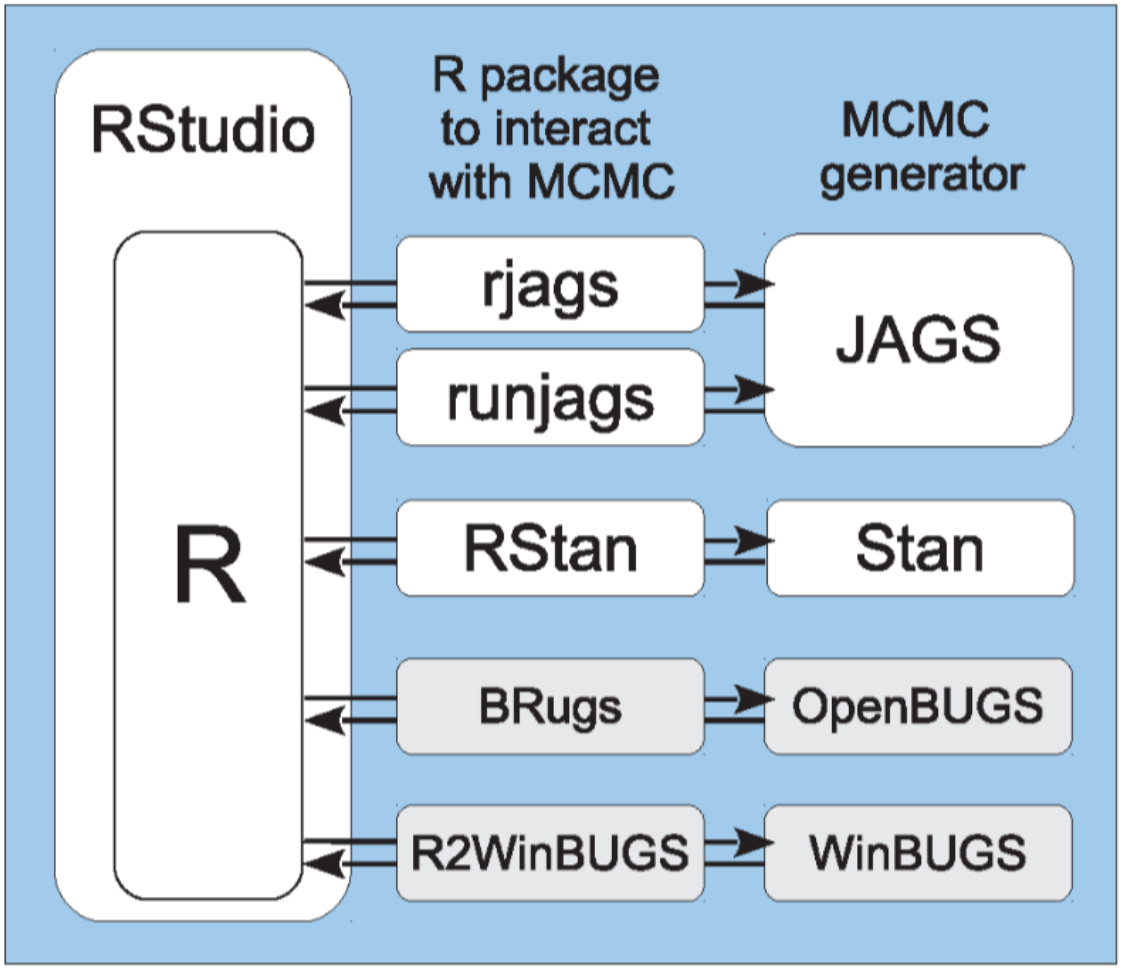
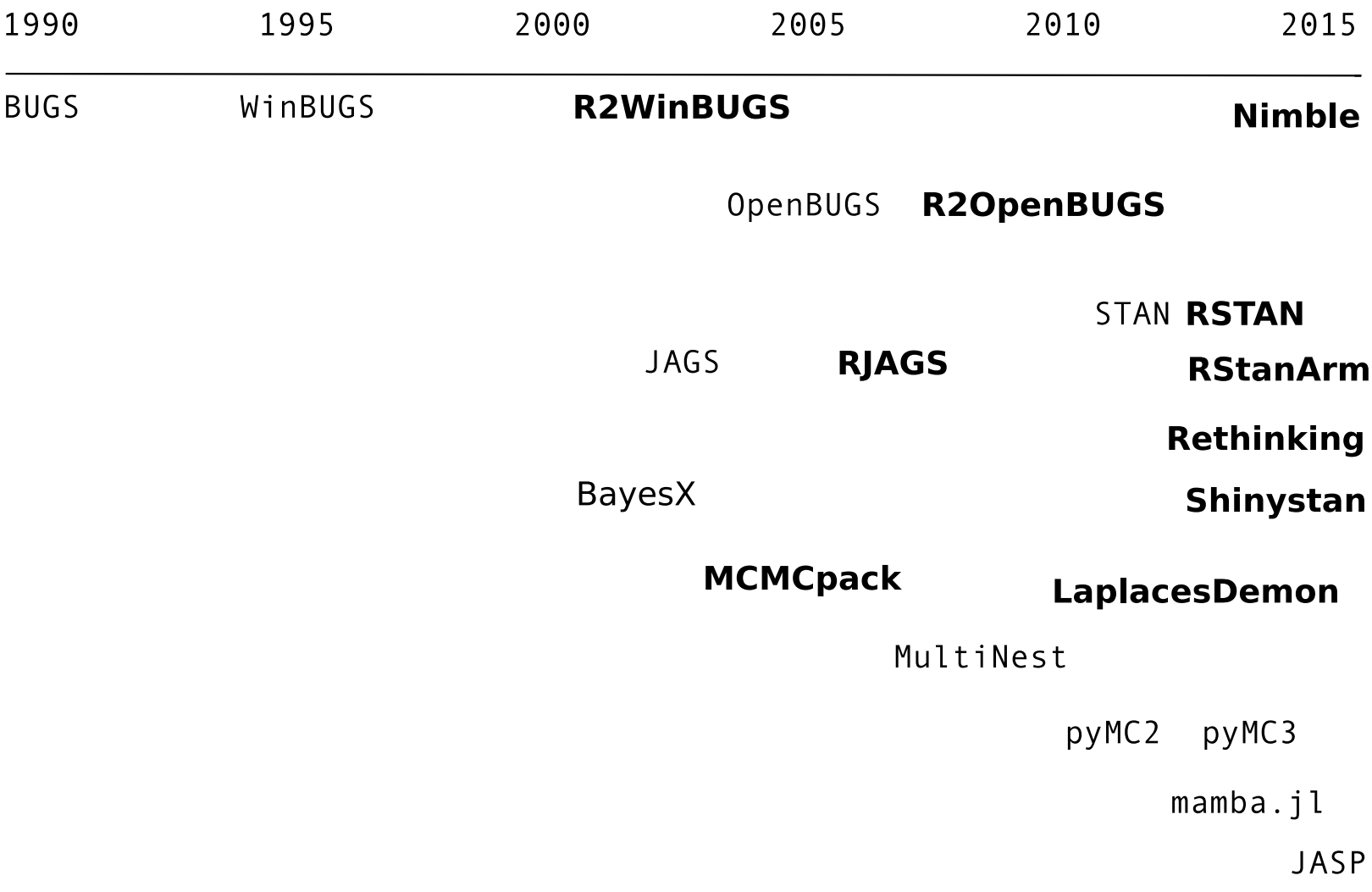


Figure 8.1: Relation of R programming language to other software tools. On the left, RStudio is an editor for interacting with R. The items on the right are various programs for generating MCMC samples of posterior distributions. The items in the middle are packages in R that interact with the MCMC generators. Copyright © Kruschke, J. K. (2014). *Doing Bayesian Data Analysis: A Tutorial with R, JAGS, and Stan. 2nd Edition.* Academic Press / Elsevier.

MCMC software for Bayesian Analysis



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.

The ways Nimble extends BUGS and JAGS

1. User-defined functions 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.

What is new in NIMBLE for statistical software?

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

NIMBLE has three components

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

Goal of NIMBLE

Make it easier to implement and apply a variety of algorithms to any model defined as a directed acyclic graph (DAG).

Possible ways to optimize performance (1/2)

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

Possible tests to optimize performance (2/2)

1. Methods for models criticism and model selection
2. Estimation of prediction error
3. “Likelihood free” or “plug-and-play”
   synthetic likelihood, approximate Bayesian computation, or iterated filtering.
4. Parametric bootstrapping
5. Test same model and algorithm with multiple data sets in one script.

Models as programming objects

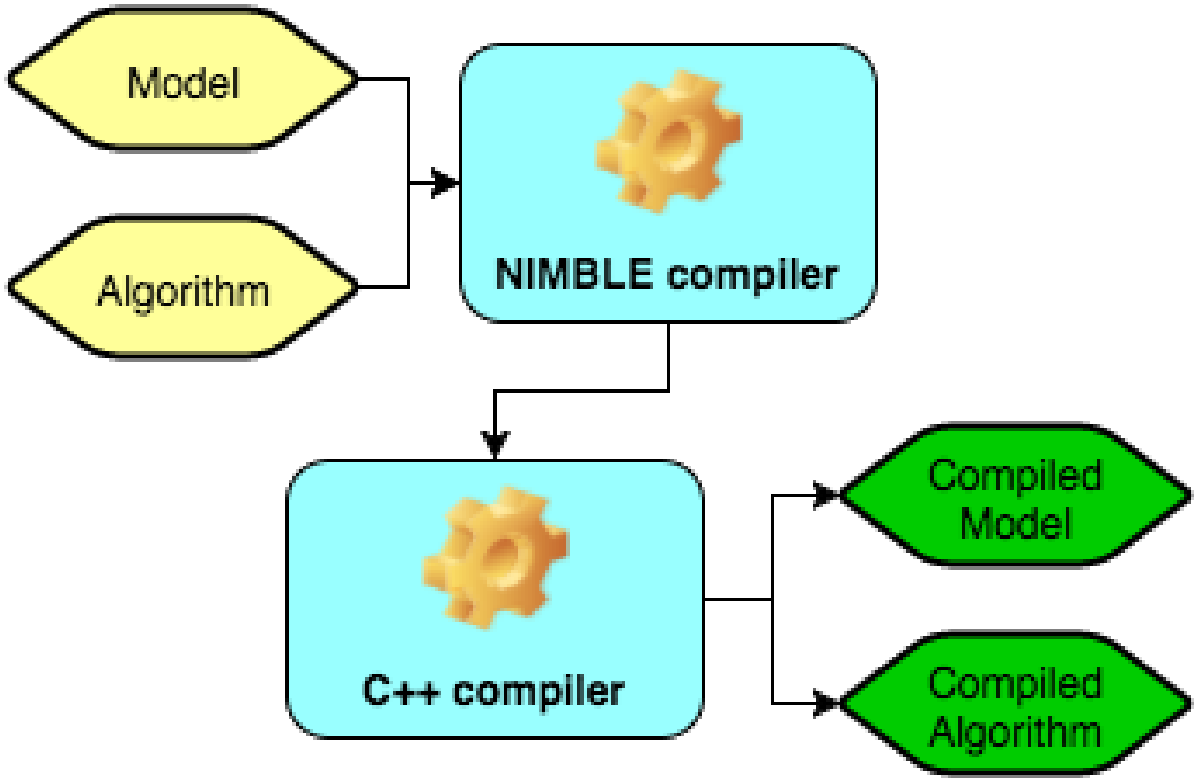


Figure: Nimble flow chart.

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

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 ones 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).
   Big improvements expected in future versions of NIMBLE (in late 2016 and 2017).

Installation

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

http://r-nimble.org/

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

Toy example

```
library(nimble)
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)
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']))
```

Example 1 output

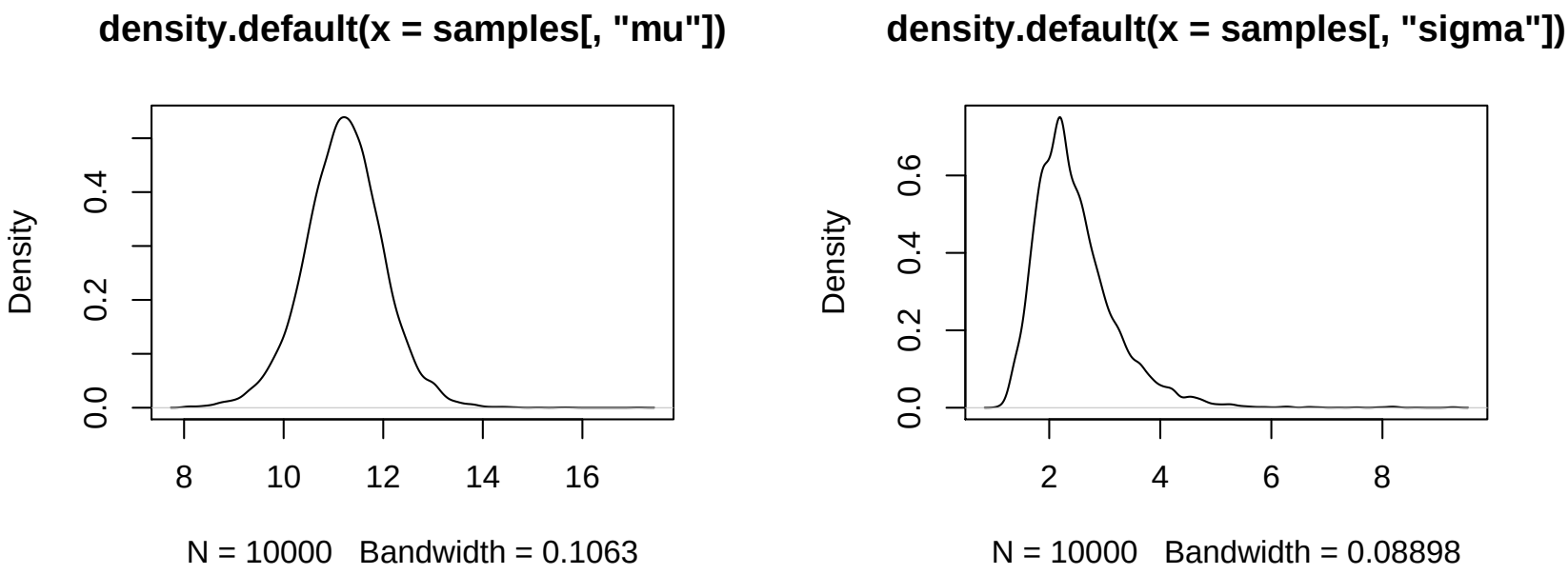
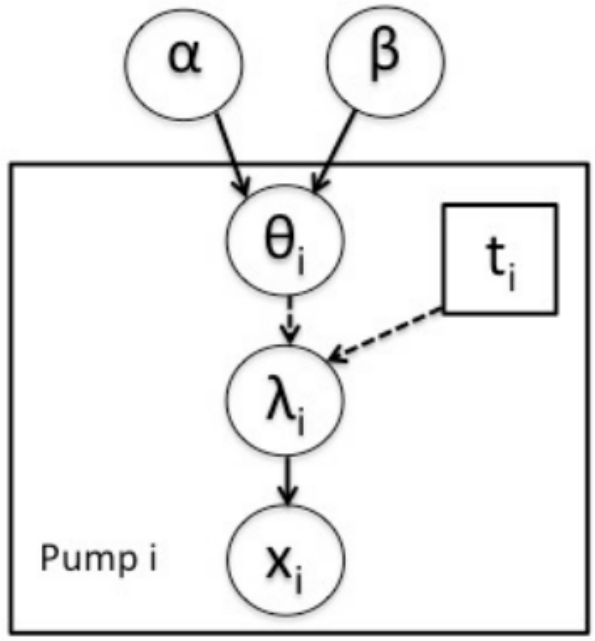


Figure: The posterior distributions of the mean and sigma.

BUGS Pump Model: 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)
```

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 for a stochastic node the log probability mass or density function, stores the resutls in an element of the corresponding log probability variable, and returns it. For deterministic nodes, calculate executes the computation, stores the result as the value of the node, and returns 0.
- **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 porbability value for a stochastic node and returns 0 for a deterministic node

modelValues objects store multiple sets of model values

- store output of MCMC
- store a set of simulated node values for input into importance sampling
- store a set of “particle” values and associated values for a particle filter

Example of workflow using the pump model

- Create the model for the pump example.
- Compile the model.
- Create a basic MCMC configuration for the pump model.
- Compile and run the MCMC
- Customize the MCMC configuration and compile and run that.
- Create, compile and run a Monte Carlo Expectation Maximization (MCEM) algorithm, which illustrates some of the exibility NIMBLE provides to combine R and NIMBLE.
- Write a short nimbleFunction to generate simulations from designated nodes of any model.

Distributions with alternative names 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

Conjugate relationships supported by NIMBLE

Prior Distribution	Sampling Distribution	Parameter
Beta	Bernoulli	**prob**
	Binomial	**prob**
	Negative Binomial	**prob**
Dirichlet	Multinomial	**prob**
Gamma	Poisson	**lambda**
	Normal	**tau**
	Lognormal	**taulog**
	Gamma	**rate**
	Exponential	**rate**
Normal	Normal	**mean**
	Lognormal	**meanlog**
Multivariate Normal	Multivariate Normal	**mean**
Wishart	Multivariate Normal	**prec**

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)