Beyond the black box: Flexible programming of hierarchical modeling algorithms for BUGS-compatible models using NIMBLE

Christopher Paciorek UC Berkeley Statistics

Joint work with:
Perry de Valpine (PI) UC Berkeley Environmental Science, Policy and Management
Daniel Turek UC Berkeley Statistics and ESPM
Cliff Anderson-Bergman Lawrence Livermore Lab (alumnus)
Duncan Temple Lang UC Davis Statistics

http://r-nimble.org

Bayesian Environmetrics Workshop
April 2016

Funded by NSF DBI-1147230
What do we want to do with hierarchical models?

1. Core algorithms
   • MCMC
   • Sequential Monte Carlo
   • Laplace approximation
   • Importance sampling

NIMBLE: extensible software for hierarchical models (r-nimble.org)
What do we want to do with hierarchical models?

1. Core algorithms
   - MCMC
   - Sequential Monte Carlo
   - Laplace approximation
   - Importance sampling

2. Different flavors of algorithms
   - Many flavors of MCMC
   - Gaussian quadrature
   - Monte Carlo expectation maximization (MCEM)
   - Kalman Filter
   - Auxiliary particle filter
   - Posterior predictive simulation
   - Posterior re-weighting
   - Data cloning
   - Bridge sampling (normalizing constants)
   - YOUR FAVORITE HERE
   - YOUR NEW IDEA HERE

NIMBLE: extensible software for hierarchical models (r-nimble.org)
What do we want to do with hierarchical models?

1. Core algorithms
   • MCMC
   • Sequential Monte Carlo
   • Laplace approximation
   • Importance sampling

2. Different flavors of algorithms
   • Many flavors of MCMC
   • Gaussian quadrature
   • Monte Carlo expectation maximization (MCEM)
   • Kalman Filter
   • Auxiliary particle filter
   • Posterior predictive simulation
   • Posterior re-weighting
   • Data cloning
   • Bridge sampling (normalizing constants)
   • YOUR FAVORITE HERE
   • YOUR NEW IDEA HERE

3. Idea combinations
   • Particle MCMC
   • Particle Filter with replenishment
   • MCMC/Laplace approximation
   • Dozens of ideas in recent JRSSB/JCGS issues

NIMBLE: extensible software for hierarchical models (r-nimble.org)
What can a practitioner do with hierarchical models?

Two basic software designs:

1. Typical R package = Model family + 1 or more algorithms
   - GLMMs: lme4, MCMCglmm
   - GAMMs: mgcv
   - spatial models: spBayes, INLA

2. Flexible model + black box algorithm
   - BUGS: WinBUGS, OpenBUGS, JAGS
   - PyMC
   - INLA
   - Stan

NIMBLE: extensible software for hierarchical models (r-nimble.org)
Existing software

Model

\[
\begin{align*}
Y(1) &\quad X(2) &\quad Y(3) \\
X(1) &\quad X(2) &\quad X(3)
\end{align*}
\]

Algorithm

e.g., BUGS (WinBUGS, OpenBUGS, JAGS), INLA, Stan, various R packages

NIMBLE: extensible software for hierarchical models (r-nimble.org)
NIMBLE: The Goal

Model

Algorithm language

NIMBLE: extensible software for hierarchical models (r-nimble.org)
Background and Goals

• Software for fitting hierarchical models has opened their use to a wide variety of communities
• Most software for fitting such models is either model-specific or algorithm-specific
• Software is often a black box and hard to extend
• Our goal is to divorce model specification from algorithm, while
  – Retaining BUGS compatibility
  – Providing a variety of standard algorithms
  – **Allowing developers to add new algorithms (including modular combination of algorithms)**
  – Allowing users to operate within R
  – Providing speed via compilation to C++, with R wrappers

NIMBLE: extensible software for hierarchical models (r-nimble.org)
NIMBLE System Summary

statistical model (BUGS code) + algorithm (nimbleFunction)

R objects + R under the hood

R objects + C++ under the hood

✧ We generate C++ code,
✧ compile and load it,
✧ provide interface object.
1. Model specification

   BUGS language → R/C++ model object

2. Algorithm specification

   NIMBLE programming language within R → R/C++
   algorithm object

3. Algorithm library

   MCMC, Particle Filter/Sequential MC, etc.
The Success of R

Programming with Data
A Guide to the S Language
### Programming with Models

You give NIMBLE:

```r
data(r = input$r)

> littersModel <- nimbleModel(littersCode, constants = list(N = 16, G = 2),
  data = list(r = input$r))
> littersModel_cpp <- compileNimble(littersModelModel) # C++ version of model
```

You get this:

```r
> littersModel$a[1] <- 5
> simulate(littersModel, 'p')
> p_deps <- littersModelModel$getDependencies('p')
> littersModelModel$calculate(p_deps)
> littersModelModel$getLogProb('r')
```

NIMBLE also extends BUGS: multiple parameterizations, named parameters, and user-defined distributions and functions.
User Experience: Specializing an Algorithm to a Model

```r
littersCode <- modelCode(
  for(j in 1:G) {
    for(l in 1:N) {
      r[i, j] ~ dbin(p[i, j], n[i, j]);
p[i, j] ~ dbeta(a[j], b[j]);
    }
    mu[j] <- a[j]/(a[j] + b[j]);
    theta[j] <- 1.0/(a[j] + b[j]);
a[j] ~ dgamma(1, 0.001);
b[j] ~ dgamma(1, 0.001);
  }

mu[j] <- a[j]/(a[j] + b[j]);
theta[j] <- 1.0/(a[j] + b[j]);
a[j] ~ dgamma(1, 0.001);
b[j] ~ dgamma(1, 0.001);
)
```

```r
> littersMCMCconf <- configureMCMC(littersModel)
> littersMCMCconf$getSamplers()
[...snip...]
[3] RW sampler; targetNode: b[1], adaptive: TRUE, adaptInterval: 200, scale: 1
[5] conjugate_beta sampler; targetNode: p[1, 1], dependents_dbin: r[1, 1]
[6] conjugate_beta sampler; targetNode: p[1, 2], dependents_dbin: r[1, 2]
[...snip...]
> littersMCMCconf$addSampler(‘a[1]’, ‘slice’, list(adaptInterval = 100))
> littersMCMCconf$addSampler(‘a[2]’, ‘slice’, list(adaptInterval = 100))
> littersMCMCconf$addMonitors(‘theta’)
> littersMCMC <- buildMCMC(littersMCMCconf)
> littersMCMC_Cpp <- compileNimble(littersMCMC, project = littersModel)
> littersMCMC_Cpp$run(20000)
```

NIMBLE: extensible software for hierarchical models (r-nimble.org)
User Experience: Specializing an Algorithm to a Model (2)

```r
littersModelCode <- quote({
  for(j in 1:G) {
    for(l in 1:N) {
      r[i, j] ~ dbin(p[i, j], n[i, j]);
      p[i, j] ~ dbeta(a[j], b[j]);
    }
    mu[j] <- a[j]/(a[j] + b[j]);
    theta[j] <- 1.0/(a[j] + b[j]);
    a[j] ~ dgamma(1, 0.001);
    b[j] ~ dgamma(1, 0.001);
  }
})

buildMCEM <- nimbleFunction(
  while(run=me(converged == 0)) {
    ...
    calculate(model, paramDepDetermNodes)
    mcmcFun(mcmc.its, initialize = FALSE)
    currentParamVals[1:nParamNodes] <- getValues(model, paramNodes)
    op <- optim(currentParamVals, objFun, maximum = TRUE)
    newParamVals <- op$maximum
    ...
  }
)
```

```r
> littersMCEM <- buildMCEM(littersModel, latentNodes = 'p', mcmcControl = list(adaptInterval = 50), boxConstraints = list(list('a', 'b'), limits = c(0, Inf)), buffer = 1e-6)
> set.seed(0)
> littersMCEM(maxit = 50, m1 = 500, m2 = 5000)
```

Modularity:

One can plug any MCMC sampler into the MCEM, with user control of the sampling strategy, in place of the default MCMC.
NIMBLE

1. Model specification
   
   BUGS language $\rightarrow$ R/C++ model object

2. Algorithm specification
   
   NIMBLE programming language within R $\rightarrow$ R/C++
   algorithm object

3. Algorithm library
   
   MCMC, Particle Filter/Sequential MC, etc.
NIMBLE: Programming With Models

We want:

• High-level processing (model structure) in R

• Low-level processing in C++
NIMBLE: Programming With Models

objectiveFunction <- nimbleFunction ( 

setup = function(model, nodes) { 
  calcNodes <- model$getDependencies(nodes) 
}, 

run = function(vals = double(1)) { 
  values(model, nodes) <<- vals 
  sumLogProb <- calculate(model, calcNodes) 
  return(sumLogProb) 
  returnType(double()) 
})

2 kinds of functions
NIMBLE: Programming With Models

objectiveFunction <- nimbleFunction (  

setup = function(model, nodes) {  
  calcNodes <- model$getDependencies(nodes)  
},

run = function(vals = double(1)) {  
  values(model, nodes) <<- vals  
  sumLogProb <- calculate(model, calcNodes)  
  return(sumLogProb)  
  returnType(double())  
})

query model structure ONCE.

NIMBLE: extensible software for hierarchical models (r-nimble.org)
NIMBLE: Programming With Models

objectiveFunction <- nimbleFunction (  

setup = function(model, nodes) {  
  calcNodes <- model$getDependencies(nodes)  
},

run = function(vals = double(1)) {  
  values(model, nodes) <<- vals  
  sumLogProb <- calculate(model, calcNodes)  
  return(sumLogProb)  
  returnType(double())  
})
The NIMBLE compiler

Feature summary:

- R-like matrix algebra (using Eigen library)
- R-like indexing (e.g. X[1:5,])
- Use of model variables and nodes
- Model calculate (logProb) and simulate functions
- Sequential integer iteration
- if-then-else, do-while
- Declare input & output types only
- Access to much of Rmath.h (e.g. distributions)
- Automatic R interface / wrapper
- Many improvements / extensions planned
How an Algorithm is Processed in NIMBLE

DSL code within nimbleFunction()

Parse in R

Parse tree of code

Process to a Reference Class in R

Abstract syntax tree

.Cpp and .h files in R TMPDIR

Writing to files from R

g++/llvm/etc.

DLL in R TMPDIR

Generation of R wrapper functions that use .Call

Access via wrappers from R

NIMBLE: extensible software for hierarchical models (r-nimble.org)
Programmer experience: Random walk updater

sampler_myRW <- nimbleFunction(contains = sampler_BASE, 

setup = function(model, mvSaved, targetNode, scale) {
  calcNodes <- model$getDependencies(targetNode)
},

run = function() {
  model_lp_initial <- getLogProb(model, calcNodes)
  proposal <- rnorm(1, model[[targetNode]], scale)
  model[[targetNode]] <<- proposal
  model_lp_proposed <- calculate(model, calcNodes)
  log_MH_ratio <- model_lp_proposed - model_lp_initial

  if(decide(log_MH_ratio)) jump <- TRUE
  else jump <- FALSE

  if(jump) {
    copy(from = model, to = mvSaved, row = 1, nodes = calcNodes, logProb = TRUE)
  } else copy(from = mvSaved, to = model, row = 1, nodes = calcNodes, logProb = TRUE)
})
1. Model specification

BUGS language $\rightarrow$ R/C++ model object

2. Algorithm specification

NIMBLE programming language within R $\rightarrow$ R/C++
algorithm object

3. Algorithm library

MCMC, Particle Filter/Sequential MC, MCEM, etc.
NIMBLE in Action: the Litters Example

Beta-binomial GLMM for clustered binary response data
Survival in two sets of 16 litters of pigs

littersModelCode <- nimbleCode(
  for(j in 1:2) {
    for(l in 1:16) {
      r[i, j] ~ dbin(p[i, j], n[i, j]);
      p[i, j] ~ dbeta(a[j], b[j]);
    }
    mu[j] <- a[j]/(a[j] + b[j]);
    theta[j] <- 1.0/(a[j] + b[j]);
    a[j] ~ dgamma(1, 0.001);
    b[j] ~ dgamma(1, 0.001);
  }
)

Challenges of the toy example:

- BUGS manual: “The estimates, particularly $a_1$, $a_2$ suffer from extremely poor convergence, limited agreement with m.l.e.’s and considerable prior sensitivity. This appears to be due primarily to the parameterisation in terms of the highly related $a_j$ and $b_j$, whereas direct sampling of $mu_j$ and $theta_j$ would be strongly preferable.”
- But that’s not all that’s going on. Consider the dependence between the p’s and their $a_j$, $b_j$ hyperparameters.
- And perhaps we want to do something other than MCMC.
Default MCMC: Gibbs + Metropolis

```r
> littersMCMCconf <- configureMCMC(littersModel, list(adaptInterval = 100))
> littersMCMC <- buildMCMC(littersMCMCconf)
> littersMCMC_cpp <- compileNIMBLE(littersModel, project = littersModel)
> littersMCMC_cpp$run(10000)
```
NIMBLE: extensible software for hierarchical models (r-nimble.org)

Red line is MLE
### Blocked MCMC: Gibbs + Blocked Metropolis

```r
> littersMCMCconf2 <- configureMCMC(littersModel, list(adaptInterval = 100))
> littersMCMCconf2$addSampler(c('a[1]', 'b[1]'), 'RW_block', list(adaptInterval = 100))
> littersMCMCconf2$addSampler(c('a[2]', 'b[2]'), 'RW_block', list(adaptInterval = 100))
> littersMCMC2 <- buildMCMC(littersMCMCconf2)
> littersMCMC2_cpp <- compileNIMBLE(littersMCMC2, project = littersModel)
> littersMCMC2_cpp$run(10000)
```
Blocked MCMC: Gibbs + Cross-level Updaters

• Cross-level dependence is a key barrier in this and many other models.
• We wrote a new “cross-level” updater function using the NIMBLE DSL.
  • Blocked Metropolis random walk on a set of hyperparameters with conditional Gibbs updates on dependent nodes (provided they are in a conjugate relationship).
  • Equivalent to (analytically) integrating the dependent (latent) nodes out of the model.

```r
> littersMCMCconf3 <- configureMCMC(littersModel, adaptInterval = 100)
> topNodes1 <- c('a[1]', 'b[1]')
> littersMCMCconf3$addSampler(topNodes1, ‘crossLevel’, list(adaptInterval = 100)
> topNodes2 <- c('a[2]', 'b[2]')
> littersMCMCconf3$addSampler(topNodes2, ‘crossLevel’, list(adaptInterval = 100)
> littersMCMC3 <- buildMCMC(littersMCMCconf3)
> littersMCMC3_cpp <- compileNIMBLE(littersMCMC3, project = littersModel)
> littersMCMC3_cpp$run(10000)
```
Stepping outside the MCMC box: maximum likelihood/empirical Bayes via MCEM

\[ \text{littersMCEM} \leftarrow \text{buildMCEM(littersModel, latentNodes = 'p')} \]
\[ \text{littersMCEM(maxit = 500, m1 = 500, m2 = 5000)} \]

- Gives estimates consistent with direct ML estimation (possible in this simple model with conjugacy for ‘p’) to 2-3 digits
- VERY slow to converge, analogous to MCMC mixing issues
- Current implementation is basic; more sophisticated treatments should help

Many algorithms are of a modular nature/combine other algorithms, e.g.
- particle MCMC
- normalizing constant algorithms
- many, many others in the literature in the last 15 years
Status of NIMBLE and Next Steps

• First release was June 2014; version 0.5 in process of being released on CRAN.
• Lots to do:
  – Sequential MC methods in next release (particle filter, ensemble Kalman filter, particle MCMC)
  – Improve the user interface and speed up compilation
  – Allow indices of vectors to be random (e.g., mixture models)
  – Refinement/extension of the DSL for algorithms
  – Additional algorithms written in NIMBLE DSL (e.g., normalizing constant calculation, Laplace approximations)
  – Advanced features (e.g., auto. differentiation, paralleliz’n)
• Interested?
  – Upcoming SBSS webinar, April 19 at 3 pm EDT: www.amstat.org/education/weblectures/index.cfm#DD
  – Announcements: nimble-announce Google site
  – User support/discussion: nimble-users Google site
  – Write an algorithm using NIMBLE!
  – Help with development of NIMBLE: email nimble.stats@gmail.com or see github.com/nimble-dev

NIMBLE: extensible software for hierarchical models (r-nimble.org)