Beyond the black box: Flexible algorithm programming for ecological models in NIMBLE

Christopher Paciorek UC Berkeley Statistics

Joint work with:

Colin Lewis-Beck	Iowa State Statistics / Google Summer of Code 2017
Perry de Valpine (PI)	UC Berkeley Environmental Science, Policy and Management
Daniel Turek	Williams College, Mathematics and Statistics
Lauren Ponisio	UC Riverside Entomology
Nick Michaud	UC Berkeley Statistics and ESPM

https://r-nimble.org

JSM Vancouver, 2018

Funded by the US National Science Foundation

What do we want to do with hierarchical models?

1. More and better MCMC
- Many different samplers
- Better adaptive algorithms

2. Numerical integration
- Laplace approximation
- Adaptive Gaussian quadrature
- Hidden Markov models

3. Maximum likelihood estimation
- Monte Carlo EM
- Data cloning
- Monte Carlo Newton-Raphson

4. Sequential Monte Carlo
- Auxiliary Particle Filter
- Ensemble Kalman Filter
- Unscented Kalman Filter

5. Normalizing constants (AIC or Bayes Factors)
- Importance sampling
- Bridge sampling
- Others

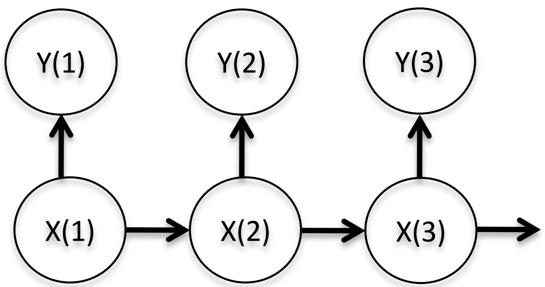
6. Model assessment
- Bootstrapping
- Calibrated posterior predictive checks
- Cross-validation
- Posterior re-weighting

7. Idea cominbations
- PF + MCMC
- Resample-move
- MCMC + Laplace/quadrature

These are just some ideas from a vast literature.

NIMBLE

Model language (BUGS/JAGS)



Algorithm Language

NIMBLE makes BUGS extensible from R:

- Add new functions
- Add new distributions
- Call external code

Goals

– Retaining BUGS compatibility
– Making BUGS more flexible
– Providing a variety of standard algorithms
– Allowing users to easily modify those 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

1. Model specification

   BUGS language ➔ R/C++ model object

2. Algorithm library

   MCMC, Particle Filter/Sequential MC, MCEM, etc.

3. Algorithm specification

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

NIMBLE's algorithm library

- MCMC samplers:
  - Conjugate, adaptive Metropolis, adaptive blocked Metropolis, slice, elliptical slice sampler, particle MCMC, specialized samplers for particular distributions (Dirichlet, CAR, Chinese Restaurant Process)
  - Flexible choice of sampler for each parameter
  - User-specified blocks of parameters
  - Cross-validation, WAIC
- Sequential Monte Carlo (particle filters)
  - Various flavors
- Write your own / easily modify ours

NIMBLE in Ecology

- User-defined distributions for integrating over high-dimensional discrete latent states
  - E.g., capture-recapture, occupancy models
- Flexibility in coding numerical tricks within a BUGS model for faster computation
- User choice of samplers and blocking
- Users can modify and add custom samplers for use in combination with NIMBLE's samplers
- Useful model selection/assessment tools:
  - WAIC
  - calibrated posterior predictive p-values
  - reversible jump

Multi-state capture-recapture: geese

- N=11,200 Canada geese
- 3 locations of 'capture' (i.e., sighting)
- 4 years of data
- 153 unique sighting histories

(survival) $\phi_r \sim \text{Uniform}(0,1)$ $\quad r = 1,2,3$

(movement) $\{\psi_{1st}, \psi_{2st}, \psi_{3st}\} \sim \text{Dirichlet}(\alpha = \{1,1,1\})$ $\quad s = 1,2,3, \quad t = 2,3,4$

(detection) $p_{rt} \sim \text{Uniform}(0,1)$ $\quad r = 1,2,3, \quad t = 1,2,3,4$

$$X_{i1} = y_{i1}$$

(site location, dead) $X_{it} \mid X_{i,t-1} \sim \text{Categorical}(p = T_t \, x_{i,t-1})$ $\quad t = 2,\ldots,k$

(site observed, not seen) $Y_{it} \mid X_{it} \sim \text{Categorical}(p = Z_t \, x_{it})$ $\quad t = 1,\ldots,k$

- Data: Armstrup et al. (2010) Handbook of Capture-Recapture Analysis
- Methods: Turek et al (2016), Env. Ecol. Stat.

Multi-state capture-recapture: filtering

- 14,437 latent variables + 21 parameters
- Discrete filtering to numerically integrate (i.e., sum) over latent variables

Filtering equations

$$P_t(x) = \Pr(X_t = x \mid y_{1:t-1})$$
$$= \sum_{x_{t-1} \in \mathcal{X}} \Pr(X_t = x \mid X_{t-1} = x_{t-1}) \Pr(X_{t-1} = x_{t-1} \mid y_{1:t-1})$$

$$Q_t(x) = \Pr(X_t = x \mid y_{1:t})$$
$$= \Pr(X_t = x \mid y_{1:t-1}) \Pr(Y_t = y_t \mid X_t = x) / \Pr(Y_t = y_t \mid y_{1:t-1})$$

$$L_t = \Pr(Y_t = y_t \mid y_{1:t-1})$$
$$= \sum_{x_t \in \mathcal{X}} \Pr(Y_t = y_t \mid X_t = x_t) \Pr(X_t = x_t \mid y_{1:t-1})$$

Matrix formulation

$$P_t = T_t \, Q_{t-1}, \qquad t \geq 2$$

$$Q_t = Z_t(y_t)' * P_t \,/\, L_t, \qquad t \geq 1$$

$$L_t = Z_t(y_t) \, P_t, \qquad t \geq 1$$

Marginalized likelihood:

$$L(\theta \mid y) = L_1 L_2 \cdots L_k$$

Multi-state capture-recapture: MCMC

– Embed filtering as a user-defined distribution in BUGS code

```
code <- nimbleCode({

   ### … priors for 'p', 'phi', 'psi' ###

   Z[1:4,1:4,1:4] <- calcZ(p[1:6])
   T[1:4,1:4,1:4] <- calcT(phi[1:3], psi[1:3,1:3,1:2])

   for (i in 1:nind) {
      y[i, first[i]:k] ~ dDHMM(length = k-first[i]+1, prior = prior[1:4], condition =
condition[1:4], Z = Z[1:k,1:k,first[i]:k], useZt = 1, T = T[1:k,1:k,first[i]:k], useTt = 1,
mult = mult[i])
   }
})
```

– 70-fold improvement in MCMC (including using weighted likelihood with unique sample histories)

Multi-state capture-recapture: MCMC (2)

Easily try out various samplers

```
conf <- configureMCMC(Rmodel)                    ## setup default MCMC samplers
conf$printSamplers()
# [1]   RW sampler: p[1]
# …
# [21] RW sampler: psi[2, 3, 2]
nodes <- Rmodel$getNodeNames(stochOnly = TRUE, includeData = FALSE)
conf$removeSamplers(nodes)                        ## remove default samplers
for(node in nodes) {
   conf$addSampler(node, type = 'slice')          ## add slice samplers
}
Rmcmc <- buildMCMC(conf)                          ## build MCMC algorithm
Cmcmc <- compileNimble(Rmcmc)                     ## compile MCMC algorithm
runMCMC(Cmcmc, 10000)                             ## run MCMC
```

Easily block parameters

```
nodes <- list(c('psi[1,1,1]','psi[2,1,1]'),       ## highly-correlated parameters (corr > 0.9)
          c('psi[1,2,1]','psi[2,2,1]'),
          c('psi[1,1,2]','psi[2,1,2]'))
for(i in seq_along(nodes)) {
   conf$removeSamplers(nodes[[i]])
   conf$addSampler(nodes[[i]], type = 'RW_block')   ## use block sampling for highly-correlated parameters
}
## build, compile and run as above
```

Multi-state capture-recapture: Results

MCMC performance aggregated across 21 parameters based on effective sample size with 10,000 iterations

Metric	Filtering (Metropolis)	Filtering + Slice	Filtering + Blocking
Minimum ESS	26	106	121
Mean ESS	294	1173	340

Multi-state capture-recapture: Results

MCMC performance aggregated across 21 parameters based on effective sample size with 10,000 iterations

Metric	Filtering (Metropolis)	Filtering + Slice	Filtering + Blocking
Minimum ESS	26	106	121
Mean ESS	294	1173	340
Minimum ESS/second	0.7	0.7	5.9
Mean ESS/second	7.8	7.6	16.7

Spatial capture-recapture: voles

- Field voles in a forest in northern England
- Data from summer 2000
- N=158 tagged voles
- Spatial grid of traps: 192 traps on 11x18 grid
- 20 observation periods
- Interest lies in understanding demographics, including survival and movement

- Model for each individual:
  - Latent state: alive or dead/emigrated at each time
  - Latent activity center at each time
  - Dispersal kernel to model movement from time to time
  - Detection and survival probabilities

Work by: Daniel Turek (NIMBLE), Torbjørn Ergon (University of Oslo)

Spatial capture-recapture: computation

Computational strategies enabled by NIMBLE:

1. Custom BUGS distribution: Integrate over latent alive/dead status via discrete filtering (see goose example).
2. Custom BUGS distribution: Move computation of dispersal into a second user-defined distribution to remove parameters and reduce model size.
3. Custom BUGS function: Carefully limit computations of "trap exposure" to avoid doing all pairwise computations of probabilities of each individual being caught in each trap.

Spatial capture-recapture: computation

2. Custom BUGS distribution: Move computation of dispersal into a second user-defined distribution to remove parameters and reduce model size

Original BUGS code:

```
for(k in first[i]:(last[i]-1)) {
   theta[i, k] ~ dunif(-3.141593, 3.141593)              # dispersal direction
   d[i, k] ~ dexp(dlambda[gr[i]])                        # dispersal distance
   S[i, 1, k+1] <- S[i, 1, k] + d[i, k] * cos(theta[i, k])   # evolution of activity center
   S[i, 2, k+1] <- S[i, 2, k] + d[i, k] * sin(theta[i, k])
}
```

Revised BUGS code:

```
for(k in first[i]:(last[i]-1)) {
   S[i, 1:2, k+1] ~ dSS(S[i, 1:2, k], dlambda[gr[i]])     # direct distribution over center
}
```

Spatial capture-recapture: computation

3. Custom BUGS functions: Carefully limit computations of "trap exposure" to avoid doing all pairwise computations of probabilities of each individual being caught in each trap.

- Original BUGS code:

```
for(k in first[i]:last[i]) {
    D[i, k, 1:R] <- sqrt((S[i, 1, k] - X[1:R, 1])^2 + (S[i, 2, k] - X[1:R, 2])^2)
    g[i, k, 1:R] <- exp(-(D[i, k, 1:R]/sigma[gr[i]])^kappa[gr[i]])  # trap exposure
    G[i, k] <- sum(g[i, k, 1:R])                                   # total trap exposure
 }
```

- Revised BUGS code:
  - Replace middle line with calls to user-defined functions that implement efficient algorithms for computing only the probabilities of an individual being trapped near to the current activity center

    ```
    g[i, k, 1:R] <- calcLocalTrapExposure(localTrapIndices, …)
    ```

  - Cache determination of nearby traps as part of model graph to limit recalculation

    ```
    localTrapIndices[i, k, 1:MaxNumberLocalTraps] <- getLocalTrapIndices(…)
    ```

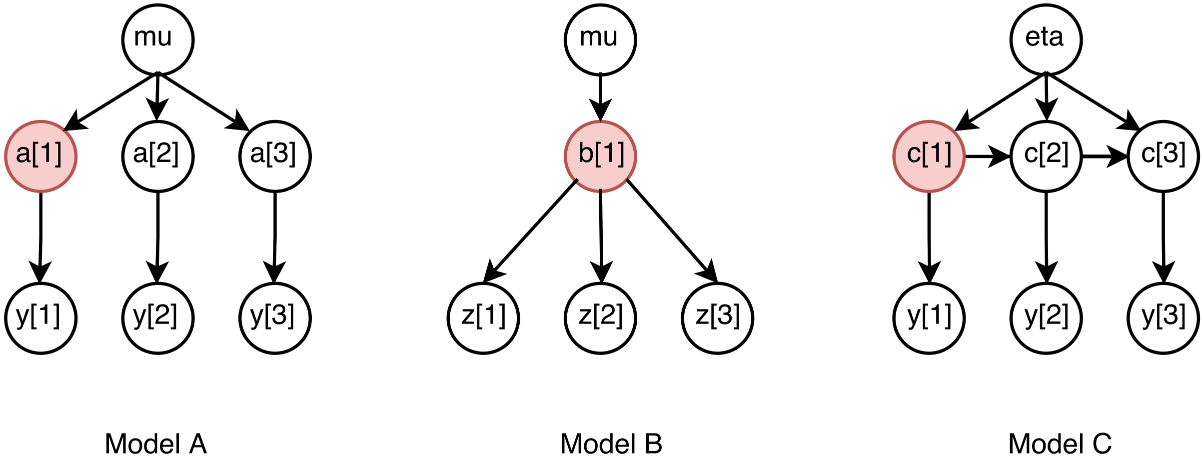
Spatial capture-recapture: computation

Time per single effectively independent sample

0. Default model (full latent state model)
   5 minutes / sample
1. Custom BUGS distribution: Integrate over latent alive/dead status via discrete filtering (see goose example).
   40 seconds / sample
2. Custom BUGS distribution: Move computation of dispersal into a second user-defined distribution to remove parameters and reduce model size.
   21 seconds / sample
3. Custom BUGS function: Carefully limit computations of “trap exposure” to avoid doing all pairwise computations of probabilities of each individual being caught in each trap.
   8 seconds per sample

Model-generic algorithm programming

Wanted: a Metropolis-Hastings sampler with normal random-walk proposals.



Challenge: It should work for any node of any model.

Solution: Two-stage evaluation.

NIMBLE: Model-generic programming

```
sampler_myRW <- nimbleFunction(

setup = function(model, mvSaved, targetNode, scale) {
   calcNodes <- model$getDependencies(targetNode)
},
run = function() {
   model_lp_initial <- calculate(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
   # .... Various bookkeeping operations ... #    })
```

2 kinds of functions

NIMBLE: Model-generic programming

```
sampler_myRW <- nimbleFunction(

setup = function(model, mvSaved, targetNode, scale) {
   calcNodes <- model$getDependencies(targetNode)
},
run = function() {
   model_lp_initial <- calculate(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
    # …. Various bookkeeping operations … #    })
```

query model structure ONCE

NIMBLE: Model-generic programming

```
sampler_myRW <- nimbleFunction(

setup = function(model, mvSaved, targetNode, scale) {
   calcNodes <- model$getDependencies(targetNode)
},
run = function() {
   model_lp_initial <- calculate(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
    # .... Various bookkeeping operations ... #    })
```

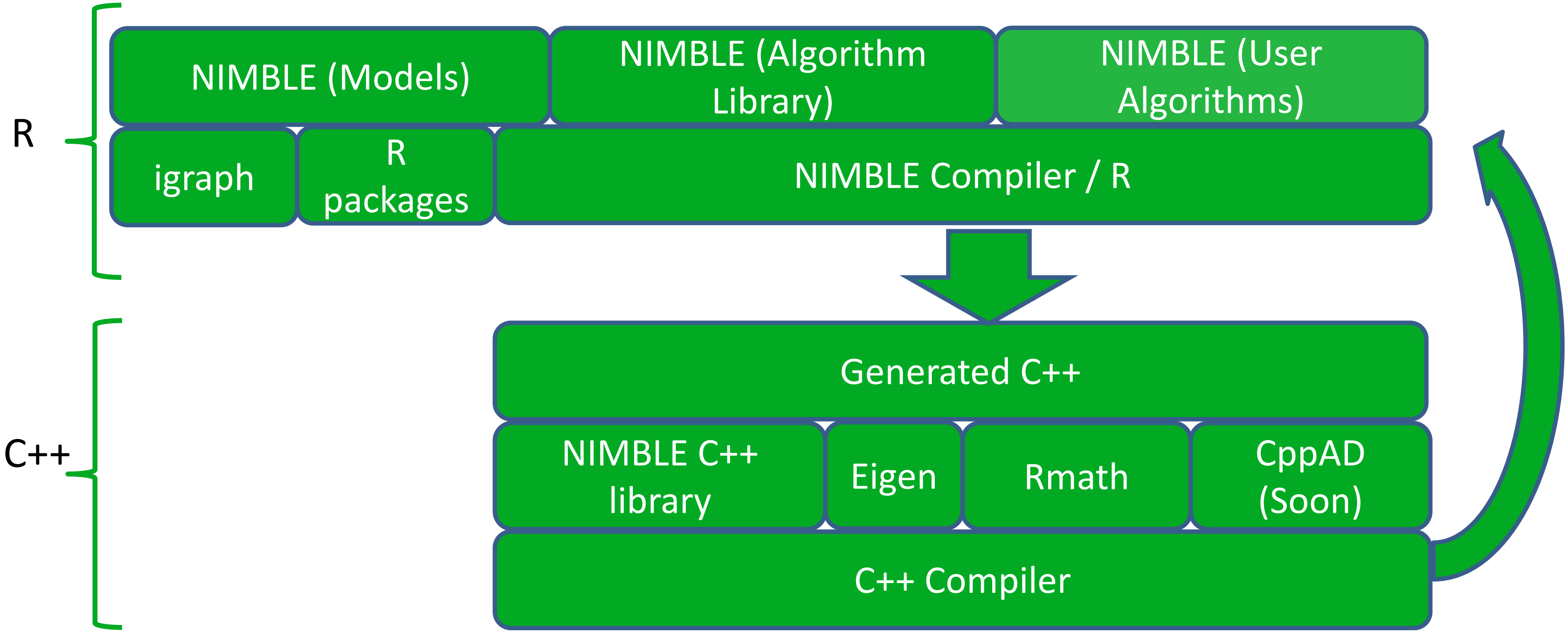
the actual (generic) algorithm

The NIMBLE compiler (run code)

<u>Feature summary:</u>

- 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
- Access to much of Rmath.h (e.g. distributions)
- Call out to your own C/C++ or back to R
- Many improvements / extensions planned
    - Derivatives (coming soon)

NIMBLE software stack



NIMBLE: What can I program?

- Your own distribution for use in a model
- Your own function for use in a model
- Your own MCMC sampler for a variable in a model
- A new MCMC sampling algorithm for general use
- A new algorithm for hierarchical models
- An algorithm that composes other existing algorithms (e.g., MCMC-SMC combinations)

NIMBLE in Ecology

- User-defined distributions for integrating over high-dimensional discrete latent states
  - To be provided in forthcoming nimbleEcology R package
- Flexibility in coding numerical tricks within a BUGS model for faster computation
- User choice of samplers and blocking
- Users can modify and add custom samplers for use in combination with NIMBLE's samplers
- Useful model selection/assessment tools: WAIC (in NIMBLE), calibrated posterior predictive p-values (nearing release), reversible jump (see r-nimble.org example)

Status of NIMBLE and Next Steps

- First release was June 2014 with regular releases since.   Lots to do:
  - Improve the user interface and speed up compilation (in progress)
  - Scalability for large models (in progress)
  - Ongoing Bayesian nonparametrics with Claudia Wehrhahn & Abel Rodriguez
  - Refinement/extension of the DSL for algorithms (in progress)
    - e.g., automatic differentiation, parallelization
  - Additional algorithms written in NIMBLE DSL
    - e.g., normalizing constant calculation, Laplace approximations, Hamiltonian MC

- Interested?
  - **We have funding for a postdoc or programmer**
  - **We have funding to bring selected users to Berkeley for intensive collaboration**
  - 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