NIMBLE: A language for algorithms for graphical models embedded in R

r-nimble.org

Perry de Valpine[1], Christopher J. Paciorek[1], Daniel Turek[2], Nick Michaud[1], & Duncan Temple Lang[3]

[1]University of California, Berkeley. [2]Williams College. [3]University of California, Davis.

Numerical
Inference for statistical
Models using
Bayesian and
Likelihood
Estimation

Introduction

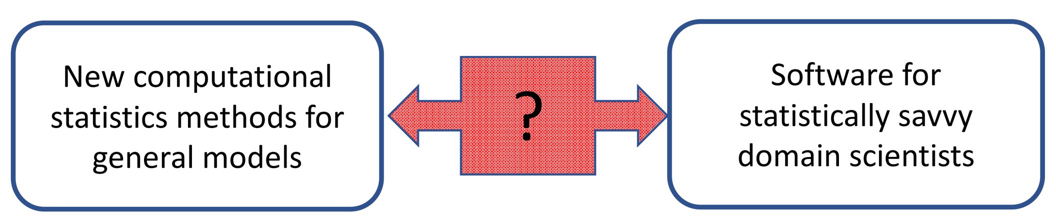
Problem-specific hierarchical statistical models (graphical models) are used by domain experts in many scientific fields:

- State-space or hidden Markov models for time-series data
- Random field models for spatial data
- Generalized linear mixed models for complex designed studies
- Capture-recapture models
- Non-parametric regression and distribution models
- Combinations of these and many more ideas

There are many unmet statistical challenges for hierarchical models:

- Efficient Markov chain Monte Carlo (MCMC) for parameter estimation
- Maximum likelihood (or empirical Bayes) estimation when the likelihood requires integration.
- Approximation of normalizing constants (likelihood or marginal likelihood) for model comparisons.
- Tools for model selection, an unresolved area of Bayesian methodology.
- Tools for model averaging.
- Tools for assessing/validating model fit or assumptions.

The gap between methods and software



Statisticians and computer scientists publish many new methods that are not accessible via software to domain scientists.

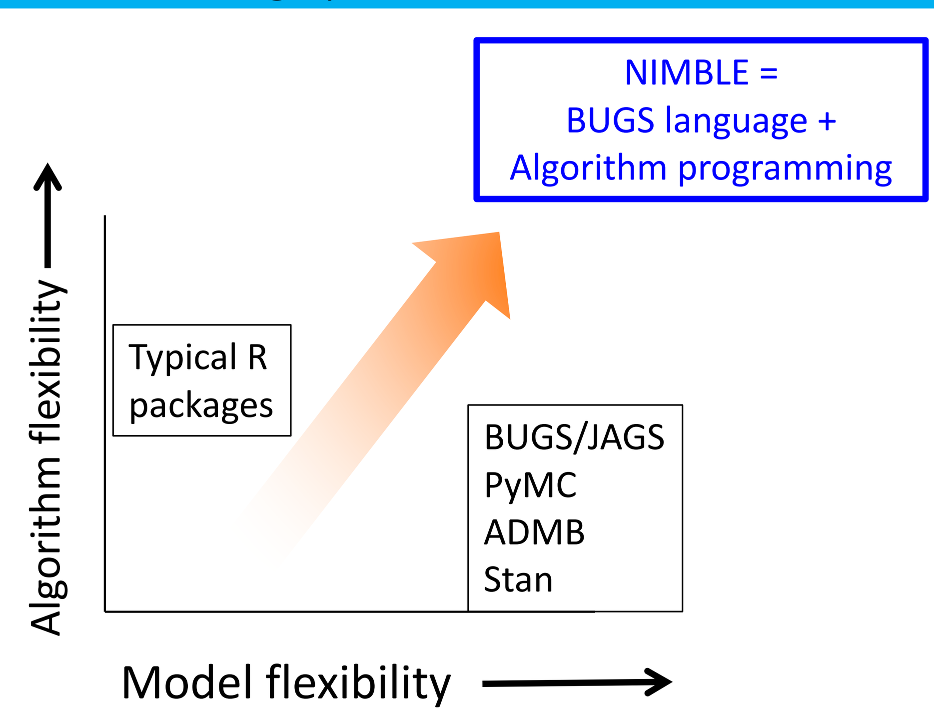
- Model-specific methods can be distributed as R packages.
- Model-generic methods are hard to code because the model must be abstracted. There has been no general system for model-generic programming.

Limitations of previous software designs

Previous software typically follows one of two basic designs:

1. Define a single model or family of models and provide specific methods for them.
   - Typical of R packages.
   - E.g. Generalized linear mixed models (GLMMS; lme4, MCMCglmm).
   - E.g. Generalized additive mixed models (mgcv).
   - E.g. Spatial models (spBayes, INLA).
   - E.g. Dirichlet process type models (dpPackage)
   - User cannot extend the model.
2. Provide a domain-specific language for writing general models and one or a few black-box algorithms.
   - E.g. MCMC is provided by BUGS (WinBUGS, OpenBUGS, JAGS), Stan, PyMC and others.
   - E.g. particle filtering is provided by BiiPS, LibBI, pomp, and others.
   - User cannot write new algorithms for the models.

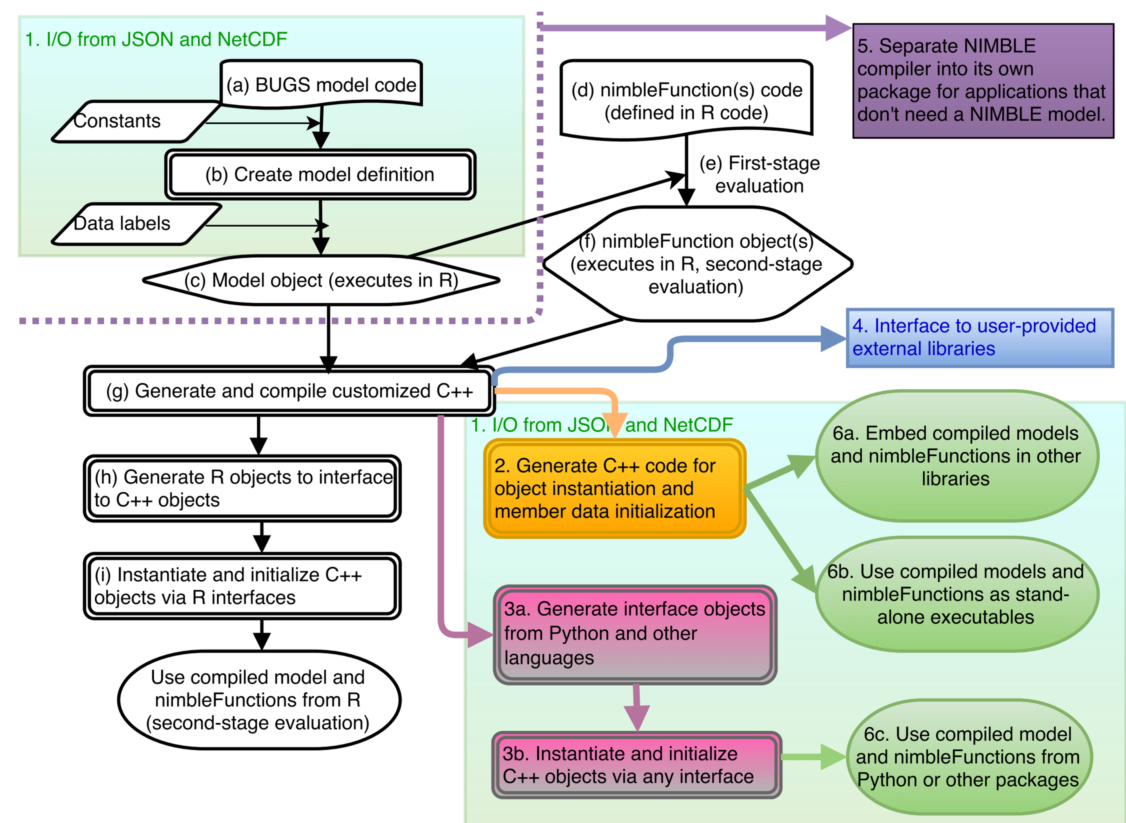
NIMBLE enables algorithm programming for general graphical models



Four components of NIMBLE

1. **Domain-specific language (DSL) for statistical models**
   - We adopt and extend the widely-used BUGS language
2. **Domain-specific language embedded within R for model-generic algorithms**
3. **Code-generator (compiler) that generates C++ from the model and algorithms DSLs.**
   - C++ objects are managed from R by dynamically-generated interface classes
4. **Algorithm library (MCMC, SMC, etc.)**

Existing and future (in colors) NIMBLE processing flows



Why R?

Benefits of embedding NIMBLE in R:

- R is widely used in applied statistics.
- The BUGS language uses extremely R-like syntax that can be natively parsed in R.
- R handles code as an object.
  - NIMBLE constructs and evaluates code for class definitions.
  - NIMBLE processes code for both BUGS and the algorithm DSL.
- NIMBLE's algorithm DSL uses two-stage evaluation, with the first stage in R and the second stage either in R (uncompiled) or C++ (compiled).
- R's packaging system (CRAN) allows users to share their own packages that use NIMBLE.

Challenges of embedding NIMBLE in R:

- R is inefficient in computation and memory use.
- NIMBLE needs some important semantic differences from R:
  - R types are dynamic, but NIMBLE types are static.
  - R passes arguments by copy, but compiled NIMBLE passes them by pointer or reference.
- NIMBLE's algorithm DSL can generally mimic familiar R behavior, but in some cases subtle differences are needed to facilitate C++ code-generation.

Features of the NIMBLE language

Compilable NIMBLE code is a narrow, enhanced subset of R designed for math and manipulation of models:

- R-style linear algebra (generated code for C++ uses the Eigen library)
- R-style math and distribution functions
- R-style flow control
- Simple class hierarchies
- *modelValues* data structure to manage many sets of model variables
  - e.g., MCMC output
- Several ways to access model variables
- Control over model operations:
  - calculate, simulate, getLogProb, calculateDiff, getParam.
- Copying of arbitrary groups of nodes between model and/or modelValues objects.
- Call out to external C++ or R code
- (beta) Derivatives of model log-density or general math (via CppAD)
- Nesting of nimbleFunctions
  - one nimbleFunction can specialize others in its setup code.

Example of programming in NIMBLE

1. Write model in BUGS code

```
pump_model_code <- nimbleCode(
 {
  for(i in 1:N) {
    theta[i] ~ dgamma(alpha,
beta)
    lambda[i] <- theta[i] * tt[i]
    x[i] ~ dpois(lambda[i])
  }
  alpha ~ dexp(1.0)
  beta ~ dgamma(2, 2)
 }
)
```

The classic "pump" example from WinBUGS.

theta[i] = random effect for pump i

tt[i] = observation duration for pump i

x[i] = observed number of pump failures

Prior distributions for random effects parameters

2. Create and compile model object in R

```
## setup of constants (N) and data (tt and x) not shown.
pump_model <- nimbleModel(pump_model_code, constants, data)
c_pump_model <- compileNimble(pump_model)
## Generates and compiles C++. Instantiates objects as needed.
```

- pump_model and c_pump_model can be used programmatically from R:
  - Access to variables
  - Access to graph structure
  - Control over simulations or calculations of any part of the model
- NIMBLE makes BUGS extensible by allowing new functions and distributions written in the algorithm language.
- These are radical departures from previous implementations of BUGS.

3. Write **model-generic algorithms** using nimbleFunctions

```
metropolis_hastings_sampler <- nimbleFunction(
  contains = sampler_BASE,

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

  run = function() {
    model_lp_current <- model$getLogProb(calcNodes)
    proposal <- rnorm(1, model[[targetNode]], scale)
    model[[targetNode]] <<- proposal
    model_lp_prop <- model$calculate(calcNodes)
    log_MH_ratio <- model_lp_prop - 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)
   })
```

First-stage evaluation queries the model to determine the Markov blanket of the targetNode and specializes an instance of the function for this case. This part runs in R.

Second-stage evaluation proposes a new value for targetNode and accepts or rejects it according to the Metropolis-Hastings acceptance rate. This part gets compiled via C++.

- This nimbleFunction illustrates writing a new MCMC sampler for a single node (vertex) of a model.
- Insertion of this nimbleFunction into a NIMBLE MCMC configuration with other samplers is not shown.
- NIMBLE comes with an adaptive Metropolis-Hastings random walk sampler. The code shown here is a simplified version without adaptation.
- This nimbleFunction is **model-generic**. Instances of this nimbleFunction can be specialized to any node in any model. Queries about the structure of a particular model are done *once* when the setup code is evaluated and then re-used in the run code.

4 . Specialize algorithms to a model, compile and run

```
mcmcConfig <- configureMCMC(pump_model)
mcmcConfig$removeSampler('beta')
mcmcConfig$addSampler('beta', 'slice')
mcmc <- buildMCMC(mcmcConfig)
C_mcmc <- compileNimble(mcmc, project =
pump_model)
runMCMC(c_mcmc, niter = 10000)
```

customize algorithm

specialize MCMC to model

execute specialized code

Implementation highlights

NIMBLE's implementation in R includes the following features:

- NIMBLE includes an R class library for representing, annotating and transforming abstract syntax trees and syntax tables of nimbleFunction classes and methods.
- NIMBLE includes an R class library for representing C++ code constructions as parse trees and symbol tables until the final step of code generation. This system could be harnessed for other uses.
- The nimbleFunction compilation process includes a modular system for processing keywords that invoke partial evaluations. For example, partial evaluation is used to resolve vectors of nodes in models at compile time, simplifying the complexity and computation time of C++ code.
- NIMBLE generates code for the Eigen linear algebra library in C++.
- (in testing) NIMBLE generates code for the CppAD auto-diff library in C++.

NIMBLE's current algorithm library

MCMC

- NIMBLE provides the most programmable, extensible MCMC system of which we are aware.
- MCMC configuration of arbitrary samplers can be programmatically created in R before specializing and compiling the nimbleFunctions.
- A variety of samplers and a default configuration system are provided.
- (beta) Samplers for Dirichlet process type nonparametric models
- Users can write new samplers and combine them with NIMBLE's samplers.

Particle Filters

- Bootstrap filter (Gordon et al. 1993. IEE-Proceedings F 140:107-113.)
- Auxiliary particle filter (Pitt and Shephard. 1999. JASA 94: 590-599).
- Liu-West filter (Liu and West. 2001. *Sequential Monte Carlo methods in practice*: 197–223. Springer.)
- Ensemble Kalman Filter
- Particle MCMC (Andrieu et al. 2010. JRSS-B 72: 269-342.)

Monte Carlo Expectation Maximization (MCEM)

- Ascent-based MCEM (Caffo et al. 2005. JRSS-B 67: 235-251)

Other

- Novel computational determination of efficient blocking schemes (Turek et al. 2016. See below.)
- (beta) Calibrated posterior predictive p-values for model assessment.

Recent progress

- Stochastic indexing for various forms of mixture models
- Enhanced NIMBLE DSL with various R-style functions
- Conditional autoregressive (CAR) spatial models (e.g., for disease mapping)
- Various improvements (speedups) to model and algorithm processing and C++ run time
- Model selection and assessment algorithms (WAIC, calibrated posterior predictive p-values [beta], cross-validation)
- Calling externally compiled code and arbitrary R code from models or nimbleFunctions.
- (in testing) automatic differentiation for model calculations and nimbleFunctions via CppAD; Langevin and HMC samplers
- (beta) Dirichlet process type models and specialized MCMC samplers
- (beta) More compact and re-usable model declarations in BUGS code.
- Various improvements to testing system

Future extensions

We plan to extend NIMBLE for:

- Parallelization, by generating code for protocols/languages starting with OpenMP/TBB and also considering Tensorflow, MPI, or CUDA.
- Greater scalability of models and algorithms.
- More compact and re-usable model declarations in BUGS code.
- Interfaces to use compiled NIMBLE models and algorithms from other languages.
- More linear algebra.
- More extensive Bayesian nonparametrics (joint work with Abel Rodriguez and Claudia Wehrhahn at UC Santa Cruz)

Publications

- P. de Valpine, D. Turek, C.J. Paciorek, C. Anderson-Bergman, D. Temple Lang, and R. Bodik. 2017. **Programming with models: writing statistical algorithms for general model structures with NIMBLE.** *Journal of Computational and Graphical Statistics*. DOI 10.1080/10618600.2016.1172487.
- D. Turek, P. de Valpine, and C.J. Paciorek. 2016. **Efficient Markov Chain Monte Carlo Sampling for Hierarchical Hidden Markov Models.** *Environmental and Ecological Statistics* 23: 549. doi:10.1007/s10651-016-0353-z.
- D. Turek, P. de Valpine, C.J. Paciorek, and C. Anderson-Bergman. 2016. **Automated Parameter Blocking for Efficient Markov Chain Monte Carlo Sampling**. *Bayesian Analysis*. doi: 10.1214/16-BA1008.

Acknowledgements

- NSF Advances in Biological Informatics (DBI-1147230)
- NSF Software Infrastructure for Sustained Innovation (ACI-155048)
- NSF DMS Collaborative Research grant (DMS-1622444)