nimble models: CHEAT SHEET

### NIMBLE workflow

- **Write model code**
  - `nimbleCode()` (NIMBLE lang)
  - `readBUGS()` (read BUGS)

- **Create model object**
  - `nimbleModel()`
  - from `nimbleCode()`
  - `constants` of the model (e.g. for-loop ranges, known index vectors)
  - `values` to label as data nodes
  - `starting values` for the parameters

- **Configure an MCMC algorithm**
  - `configureMCMC()`
  - Advanced customization

- **Build an MCMC object**
  - `buildMCMC()`
  - model object or MCMC configuration variables names for MCMC output
  - thinning interval

- **Run MCMC**
  - `runMCMC()` using MCMC object `nimbleMCMC` one-line invocation
  - Compile in C++ for faster execution
  - MCMC from `buildMCMC()` compiled or uncompiled
  - number of MCMC iterations & burnin

### Writing model code

#### Split code over multiple lines to help people read it.

```nimble
modelCode <- nimbleCode({
  beta0 ~ dnorm(0, sd = 1000)
  beta1 ~ dnorm(0, sd = 100)
  fixed_effects[1:N] <- beta0 + beta1 +
    for(i in 1:N) { 
      log(eta[i]) <- fixed_effects[i] +
        y[i] ~ dpois(eta[i])
    }
  for(iGroup in 1:numGroups) {
    alpha[iGroup] ~ dnorm(0, sd = sdGroups)
  }
})
```

#### Use named arguments for non-default parameterization
- `e.g. beta0 and beta1 follow equivalent distributions (default is precision, tau)`

#### Link functions can be declared on the left-hand side.

#### Order of declaration does not matter
- `alpha[iGroup]` can be declared after being used in other declarations.

#### Models can access and change log-probabilities.

```nimble
model$calculate(nodes) returns sum of log probability densities.
model$calculateDiff(nodes) returns difference in sum of log probability densities between current and previous node values.
model$calculateLogProb(nodes) returns sum of most recently calculated log probability densities.
model$simulate(nodes, includeData = FALSE) simulates into stochastic nodes.
includeData = FALSE protects data.
```

#### Models know about nodes, variables and relationships.

```nimble
model$getNodeNames() returns node names e.g. "y", "y[1]", "y[2]", ... model$getVarNames() returns variable names e.g. "eta"
model$expandNodeNames(nodes) e.g. "y" is expanded to "y[1]", "y[2]", ...
model$getDependencies(nodes, ...) returns nodes that depend on input nodes.
```

#### Models know properties of nodes.

```nimble
model$dimenion(node) model$distribution(nodes) model$isDeterministic(nodes) model$isStoch(nodes) model$data(nodes) model$isDiscrete(nodes) model$multiVarate(nodes) model$binary(nodes) model$endNode(nodes) model$truncated(nodes)
```

#### Models are graphs

- `myModel$plot()`

#### Uncompiled models can be debugged, updated, and copied.

- Flag nodes as data and set inits
- `myModel$setData("y")` `myModel$setInits(c(1, 2))`

- Debug model errors
- `myModel$check()` check for missing/invalid values.
- `myModel$initializeInfo()` which nodes are not fully initialized?
- `myModel$checkBasics()` check for size/dimension mismatches and NA.
- Make a copy
- `myModel$newModel(replicate = TRUE)`

#### Vectorized declarations
- `create vector nodes. This means fixed_effects[1:N] will be a single node. One vector node vs. multiple scalar nodes give different model graphs, so use with care.`

#### Provide explicit index ranges or use empty brackets `()`

#### Nested indexing is a good way to implement experimental groups or factor levels. If groups are known from the design, include them in constants.

---

CC BY SA NIMBLE Development Team • nimble.stats@gmail.com • See r-nimble.org for user manual, user group, and examples • package version 0.9.1 • Updated: 2020-05
nimble distributions and functions: CHEAT SHEET

Declarations

STOCHASTIC
x ~ ddist(args)

DETERMINISTIC
z <- fn(args)

TRUNCATED STOCHASTIC
x ~ T(ddist(args), min, max)

CENSORED STOCHASTIC
seg ~ dinterval(t, c[1:nSegments])
t ~ ddist(args)

CONSTRAINT
one ~ dconstraint(condition)

Deterministic Functions

SCALAR or COMPONENT-WISE
Logical: |, &, !, >, >=, <, <=, !=, ==, equals, step
Arithmetic: +, -, *, /, %, %%, ^, pow(x, y)
Trigonometric: sin, cos, tan, asin, acos, atan, asinh, acosh, atanh
Links: logit, probit, cloglog (links can also be used on left-hand side of a declaration)
Inverse links: ilogit/expit, logit/exp, iprobit/phi, icloglog
Rounding: ceiling, floor, round, trunc
Specials: lgamma/loggam, besselK, besselJ, besselI, besselY, logfact, logfactgamma
Distributions: d, p, q, r forms of available distributions can be used as deterministic functions.

VECTOR and/or MATRIX
Returning scalar: inprod, logdet, sum, mean, sd, prod, min, max
Returning vector: pmin, pmax, eigen(x)$vals, svd(x)$d
Returning matrix: inverse, chol, %%, t, solve, forwardsolve, backsolve, eigen(x)$vectors, svd(x)$u, svd(x)$v

Multivariate distributions

DIRICHLET
y[] ~ ddirch(alpha[])

MULTINOMIAL
y[] ~ dmulti(prob[], size)

MULTIVARIATE NORMAL
y[] ~ dnorm(mean[, scale[, prec]], df, scale_param)

MULTIVARIATE STUDENT T
y[] ~ dt(mu[, scale[, prec]], df, prec_param)

WISHART
y[,] ~ dwish([R[,] | S[,] | cholesky[,]], df[, scale_param])

INVERSE WISHART
y[,] ~ dinwish([S[,] | R[,] | cholesky[,]], df[, scale_param])

CHINESE RESTAURANT PROCESS
y[] ~ dCRP[conc, size]

Conditional autoregressive (improper)
y[] ~ dcar_normal(adj[], weights[], num[], tau, c[, zero_mean])

Proper
y[] ~ dcar_proper(mu[], C[], adj[], num[], M[, tau, gamma])

Bayesian nonparametric distributions

Distributions for spatial models

CONDITIONAL AUTOREGRESSIVE
 intrinsic (improper)
y[] ~ dcar_normal(adj[], weights[], num[], tau, c[, zero_mean])

Proper
y[] ~ dcar_proper(mu[], C[, adj], num[], M[, tau, gamma])

STICK BREAKING PROCESS
y[] ~ stick_breaking(z[])
z = vector of breaking points

Write you own!

NIMBLE allows you to write new distributions and functions using nimbleFunction().