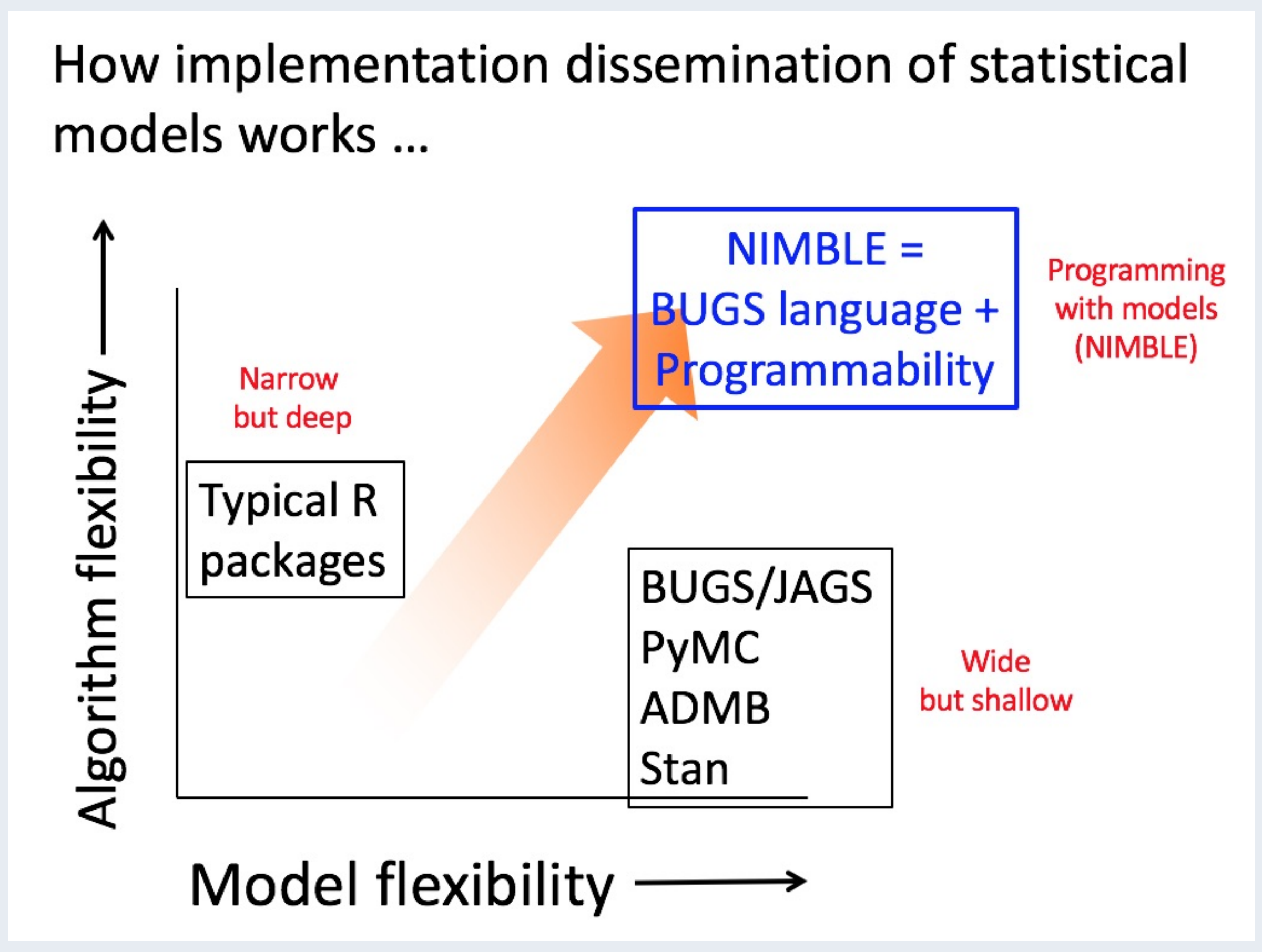
The NIMBLE Environment for Statistical Computing

Claudia Wehrhahn[1], Abel Rodriguez[1], and Christopher Paciorek[2]
[1] Applied Mathematics and Statistics, UC Santa Cruz, USA.
[2] Department of Statistics, UC Berkeley, USA.

Introduction

NIMBLE is an open source system within the statistical R software environment. It is a flexible system that allows users to define high-level hierarchical models and select and/or define algorithms for model sampling at each level of the model. Additionally, computations are speedup up through C++ compilations.
Bayesian non parametric (BNP) models are a very flexible approach for statistical modeling, yet their, usually difficult, implementation requires the development of a statistical software allowing more users to enjoy the richness of the BNP modeling approach.
NIMBLE can be downloaded from `https://r-nimble.org`. Several examples, the manual, links for questions, and much more, are available from the web page!

The NIMBLE environment


How implementation dissemination of statistical models works ...
NIMBLE = BUGS language + Programmability
Programming with models (NIMBLE)
Narrow but deep
Typical R packages
BUGS/JAGS
PyMC
ADMB
Stan
Wide but shallow
Algorithm flexibility
Model flexibility

The NIMBLE environment involves three components:

- A declarative language for statistical model specification using BUGS language from `R`. You can write your model using BUGS code and turn it into a model object that can be used for any algorithm: provided by NIMBLE or your own. Also, BUGS code is extended to include multiple parametrizations.

- A new domain-specific language for programming computational statistical algorithms. You can write your own algorithm using `nimbleFunction()` or use one from the NIMBLE library. Algorithms provided by NIMBLE can be customized.

- A compiler that generates C++ code and compiles it. You can use this compiler for speeding up computations in the BUGS model, self-defined algorithm, or any numerical computation, without knowing any C programming!

NIMBLE: BNP modeling

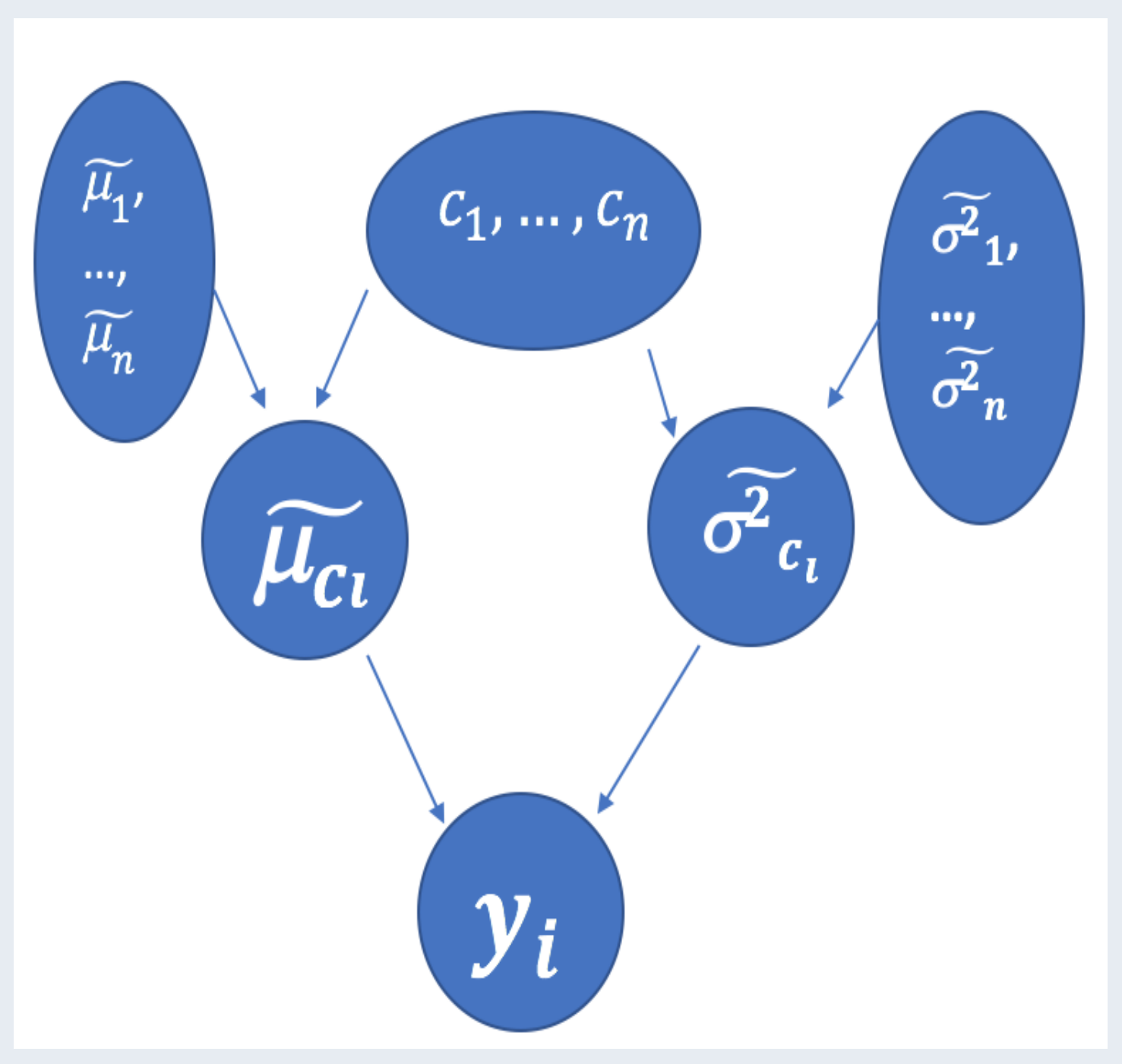
NIMBLE provides support for Bayesian non-parametric (BNP) mixture modeling. The current implementation provides support for hierarchical specifications involving Dirichlet process (DP) mixtures A DP mixture model takes the form $y_i \mid G \overset{iid}{\sim} \int h(y_i \mid \theta)G(d\theta), \quad G \mid \alpha, G_0 \sim DP(\alpha, G_0)$.

As an example, consider a machine assembling hard drives, which sometimes end up being defective. The time (in days) between two consecutive defective hard drives are defectively assembled is recorded, in the log scale. The aim is to "find" the density function that describes the time between two consecutive defective hard drives are found. To this aim, we consider $y_i \mid \mu, \sigma^2 \sim N(\mu, \sigma^2)$ and $G_0 \equiv N(0, 100) \times IG(1, 1)$.

When the random measure G is integrated out, then the model can be written using latent variables c_i, taking the form

$$\begin{aligned} \alpha &\sim Gamma(1,1), \\ \boldsymbol{c} \mid \alpha &\sim CRP(\alpha), \\ \tilde{\mu}_i &\overset{iid}{\sim} N(0,100), \\ \tilde{\sigma}_i^2 &\overset{iid}{\sim} IG(1,1), \\ y_i \mid \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\sigma}}^2, c_i &\overset{ind}{\sim} N(\tilde{\mu}_{c_i}, \tilde{\sigma}^2_{c_i}). \end{aligned}$$

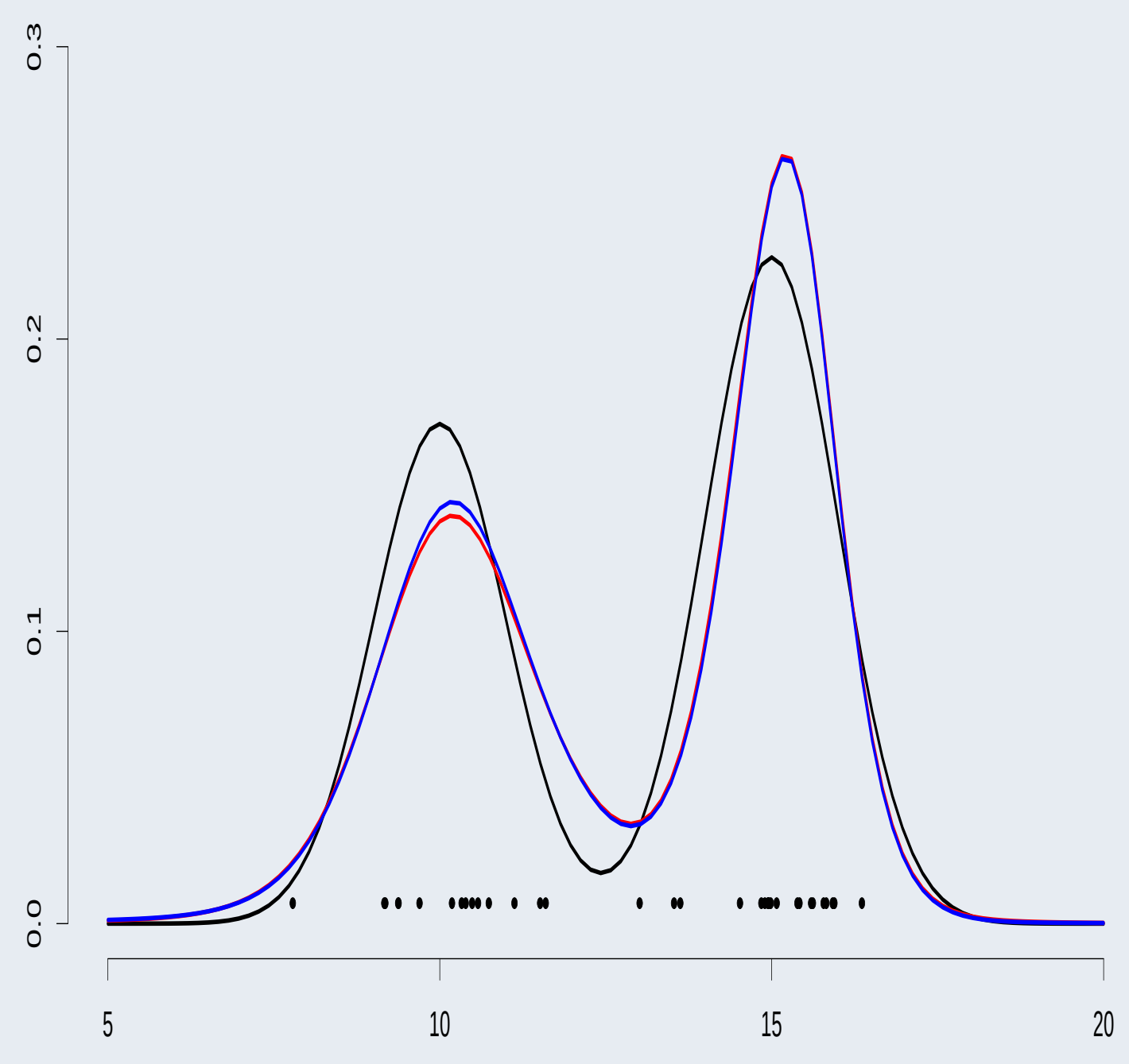
```
Code <- nimbleCode( {
    alpha ~ dgamma(1, 1)
    c[1:n] ~ dCRP(alpha, size=n)
    for(i in 1:n){
      mutilde[i] ~ dnorm(0, var = 100)
      s2tilde[i] ~ dinvgamma(1, 1)
      y[i] ~ dnorm(mutilde[c[i]],var=s2tilde[c[i]])
    }
 })
```


$\tilde{\mu}_1, \dots, \tilde{\mu}_n$
$c_1, \dots, c_n$
$\tilde{\sigma}^2_1, \dots, \tilde{\sigma}^2_n$
$\tilde{\mu}_{c_i}$
$\tilde{\sigma}^2_{c_i}$
y_i

If a stick-breaking representation is assumed for the random measure G, where $w_l = v_l \prod_{m<l}(1 - v_m)$, $l = 1, \dots, L$, then the model takes the form

$$\begin{aligned} \alpha &\sim Gamma(1,1), \\ v_l \mid \alpha &\overset{iid}{\sim} Beta(1,\alpha), \\ \mu_l^* &\overset{iid}{\sim} N(0,100), \\ \sigma^{*2}_l &\overset{iid}{\sim} IG(1,1), \\ \\ z_i \mid \boldsymbol{w} &\overset{iid}{\sim} Cat(\boldsymbol{w}), \\ y_i \mid \boldsymbol{\mu}^*, \boldsymbol{\sigma}^{*2}, c_i &\overset{ind}{\sim} N(\mu^*_{c_i}, {\sigma^*}^2_{c_i}). \end{aligned}$$

```
Code <- nimbleCode( {
    alpha ~ dgamma(1, 1)
    for(l in 1:24)
      v[l] ~ dbeta(1, alpha)
    for(l in 1:25) {
      mustar[l] ~ dnorm(mean=0, var=100)
      s2star[l] ~ dinvgamma(1, 1) }
    for(i in 1:n){
      z[i] ~ dcat(stick_breaking(v[1:24]))
      y[i] ~ dnorm(mustar[z[i]], var=s2star[z[i]]) }
})
```



The NIMBLE model is defined as `model=nimbleModel(Code, ...)` and computations are seeded up compiling the model in C++ as `Cmodel=compileNimble(model)`.

NIMBLE: model sampling

- Standard code lines for sampling posterior distributions: `mConf=configureMCMC(model); mMCMC=buildMCMC(mConf); CmMCMC=compileNimble(mMCMC); CmMCMC$run(1000)`.
- Specialized algorithms for BNP models: `CRP_concentration sampler: alpha; CRP sampler: c[1:35]; conjugate_dbeta_dcat sampler: v[1]`.
- Samplers can be customized at any level of the model. For instance, `modelConf$removeSamplers('mutilde')` and `modelConf$addSampler('mutilde', type="RW_block")`.
- User-defined samplers can be defined and can be included in the MCMC steps.