Nimble for Bayesian Disease Mapping

Andrew Lawson
Medical University of South Carolina(MUSC)
and Usher Institute, University of Edinburgh

Nimble

- Nimble is an R package that allows the programming of Bayesian models directly, as per WinBUGS/OpenBUGS or JAGS
- It parses the model code written, and writes its own samplers in C++
- It provides great flexibility in modelling and considerable speed ups compared to BUGS
- It uses a range of samplers (random walk MH; slice)
- It provides MCMC, sequential MC (particle filters) and MCEM facilities

Package 'nimble'

June 2, 2019

Title MCMC, Particle Filtering, and Programmable Hierarchical Modeling

Description A system for writing hierarchical statistical models largely compatible with 'BUGS' and 'JAGS', writing nimbleFunctions to operate models and do basic R-style math, and compiling both models and nimbleFunctions via custom-generated C++. 'NIMBLE' includes default methods for MCMC, particle filtering, Monte Carlo Expectation Maximization, and some other tools. The nimbleFunction system makes it easy to do things like implement new MCMC samplers from R, customize the assignment of samplers to different parts of a model from R, and compile the new samplers automatically via C++ alongside the samplers 'NIMBLE' provides. 'NIMBLE' extends the 'BUGS'/'JAGS' language by making it extensible: New distributions and functions can be added, including as calls to external compiled code. Although most people think of MCMC as the main goal of the 'BUGS'/'JAGS' language for writing models, one can use 'NIMBLE' for writing arbitrary other kinds of model-generic algorithms as well. A full User Manual is available at <https://r-nimble.org>.

Version 0.8.0

More Features

- Nimble allows extension of classic BUGS/JAGS code to include
  - R functions via `nimbleRcall`
  - User defined distributions and functions
  - New MCMC samplers
  - Can call external compiled code
- Run completely in R so there is no need to call external programs
- Executes faster than conventional OpenBUGS code

Some Win/OpenBUGS basics

- WinBUGS is a Bayesian hierarchical modeling package
- It uses code that defines a
  - Data model (via likelihood)
  - Prior distributions
- It is a windows package and can be run via Wine emulator on MACs
- It can be called from R using R2WInBUGS
- OpenBUGS can be called from R using BRugs

Specifying a model

Notation and data model

$y_i \sim Pois(e_i \theta_i)$ *data model*

y_i *incident count in i th area*

e_i *expected count/rate*

θ_i *relative risk*

e_i *known (computed from a standard population)*

θ_i *modelled risk*

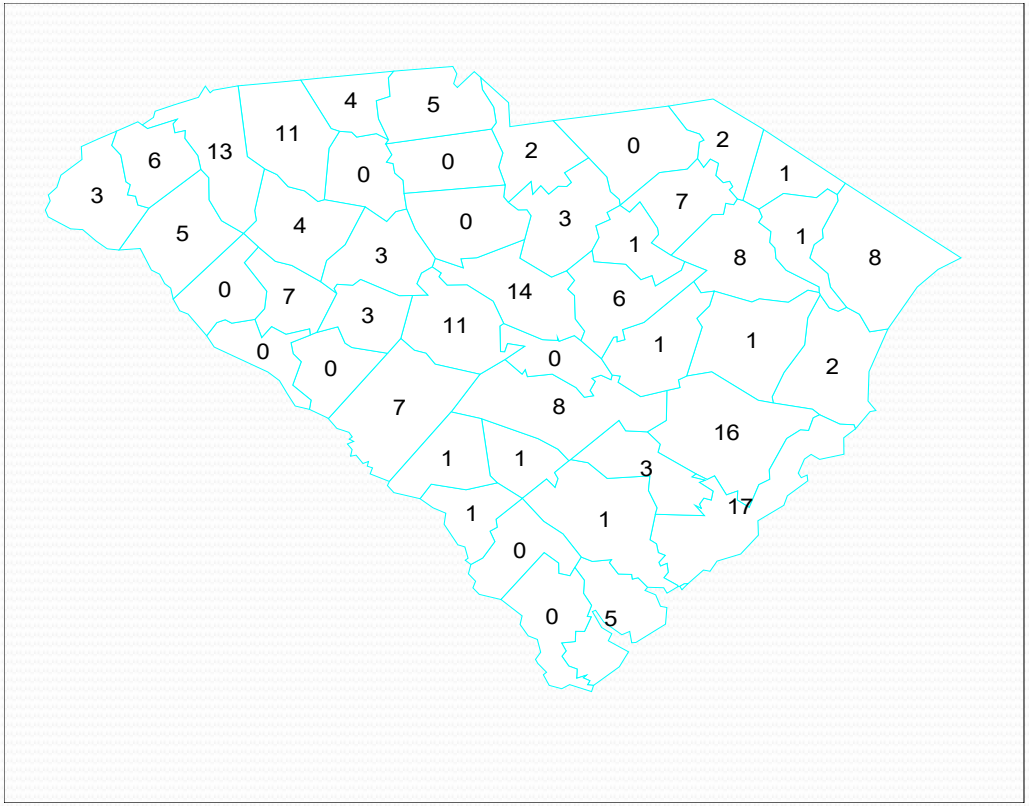
Gamma-Poisson model

- Assumes a Poisson data model and gamma prior for the relative risk
- Additional prior distributions assumed for gamma parameters

$$y_i \sim Pois(e_i\theta_i)$$
$$\theta_i \sim Gamma(a,b)$$
$$a \sim Exp(0.1) \qquad \# \textit{ exponential distribution}$$
$$b \sim Exp(0.1)$$

A WinBUGS example

- South Carolina congenital deaths 1990



WinBUGS Code for Poisson-Gamma Model

```
model
{
for (i in 1:m)
{
    # Poisson likelihood for observed counts
    y[i]~dpois(mu[i])
    mu[i]<-e[i]*theta[i]
    # Relative Risk
    theta[i]~dgamma(a,b)
}

# Prior distributions for "population" parameters
a~dexp(10)
b~dexp(10)

}
```

- Poisson likelihood/data model
- Gamma (a,b) prior
- Exponential priors for a and b

Data Entry

list(

m=46,

y=c(0,7,1,5,1,1,5,16,0,17,4,0,0,1,1,7,1,3,0,0,8,2,13,7,0,8,0,3,2,4,1,11,0,1,2,3,3,8,6,14,3,11,
6,0,1,5),

e=c(1.129778827,6.667008775,0.650279674,6.988864371,0.95571406,1.123210345,5
.908349156,8.539026017,0.601016062,18.92051111,2.272694617,1.73736337,2.019
808077,1.688099759,1.747216093,3.221840201,1.835890594,5.221942834,0.9787
03751,1.254579976,6.407553754,2.676656232,16.57884744,3.077333607,1.08708
3697,7.606301637,1.018114641,2.15774619,2.844152512,2.955816698,0.985272233
,9.22871658,0.38097193,1.855596038,1.579719813,1.579719813,2.647098065,4.79
1707292,4.144711859,15.70852363,0.765228101,11.32077795,6.256478678,1.50089
8035,2.085492893, 7.297583004))

WinBUGS14

File Tools Edit Attributes Info Model Inference Options Doodle Map Window Help

Gamma_Poisson

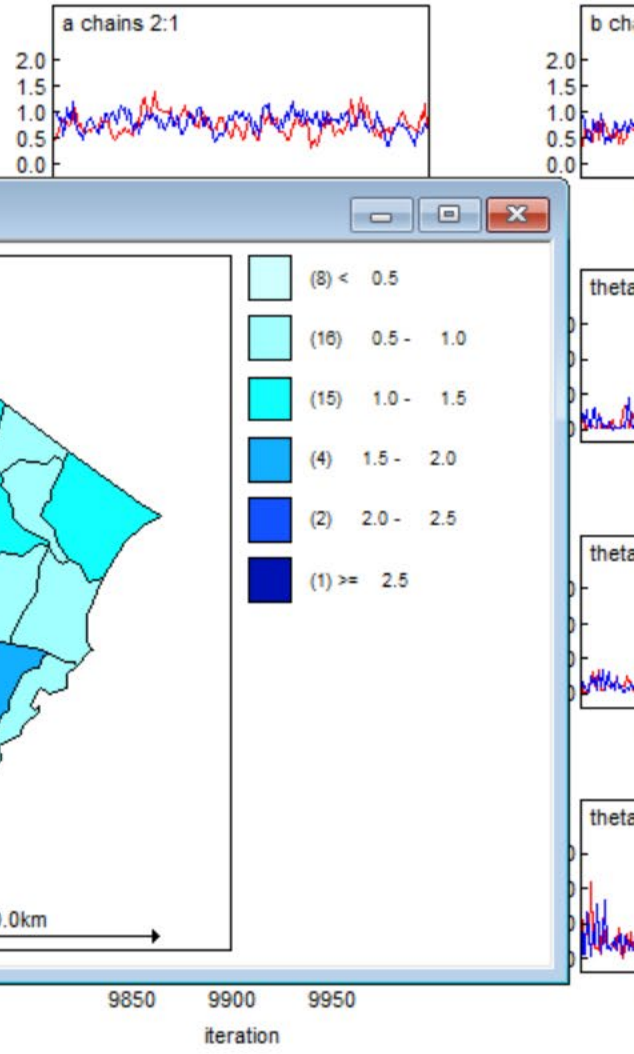
```
model;
{
   for( i in 1 : m ) {
      y[i] ~ dpois(mu[i])
      mu[i]<-expe[i]*theta[i]
theta[i] ~ dgamma(a,b)
   }
   a ~ dexp( 10)
   b ~ dexp( 10)
}
#####################################
## DATA##  SC congen 1990 incidence SC countie

list(m=46,y=c(0,7,1,5,1,1,5,16,0,17,4,0,0,1,1,7,1,3,0
expe=c(1.0807,6.3775,0.622,6.6854,0.9142,1.0744
1.6713,3.0819,1.7562,4.9952,0.9362,1.2001,6.1293
8.828,0.3644,1.775,1.5111,1.5111,2.5321,4.5836,3


#####################################
## inits ############################

list(theta=c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
1, 1, 1, 1, 1), a=10,b=10)
```

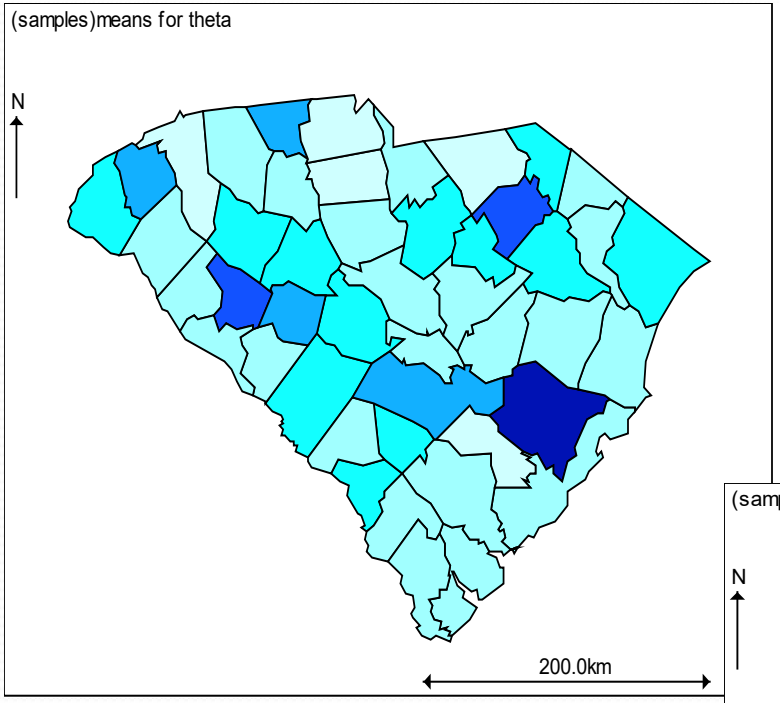
Dynamic trace

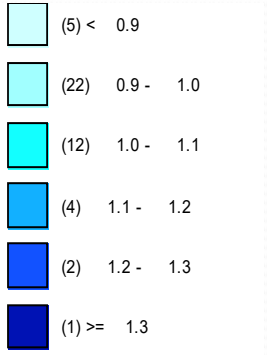


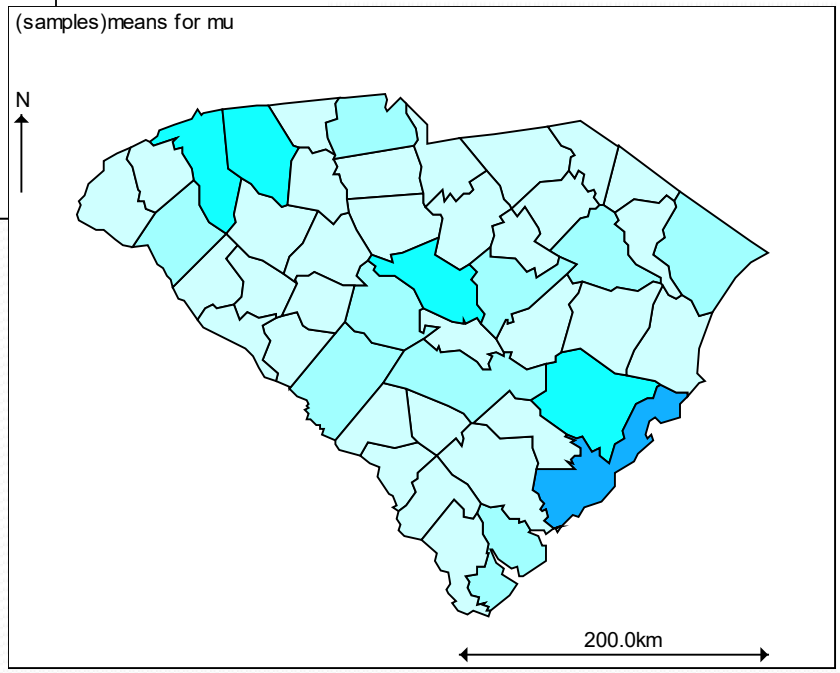
(samples)means for theta

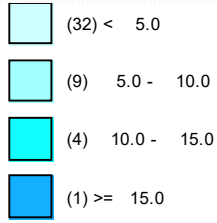
Type here to search

Relative risk and mean maps









Posterior Summary Statistics

- a small sample of summary statistics available (a, b, and first 10 regions for theta: mean, sd, MC error 2.5%,median, 95% CI)

Node statistics

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
a	16.13	6.184	0.5473	6.429	15.71	29.17	10001	10000
b	16.01	6.195	0.5483	6.386	15.59	29.62	10001	10000
theta[1]	0.9392	0.2624	0.004627	0.4738	0.9261	1.511	10001	10000
theta[2]	1.024	0.2313	0.002664	0.6236	1.006	1.529	10001	10000
theta[3]	1.034	0.2839	0.003567	0.5578	1.008	1.674	10001	10000
theta[4]	0.9111	0.2119	0.003457	0.5321	0.8986	1.364	10001	10000
theta[5]	1.02	0.2802	0.003969	0.5459	0.9956	1.638	10001	10000
theta[6]	1.002	0.2706	0.003371	0.5391	0.9766	1.606	10001	10000
theta[7]	0.9624	0.2277	0.002804	0.5637	0.9443	1.459	10001	10000
theta[8]	1.324	0.2608	0.00755	0.8846	1.301	1.913	10001	10000
theta[9]	0.9715	0.276	0.003904	0.4931	0.9477	1.582	10001	10000
theta[10]	0.9474	0.1705	0.002318	0.6418	0.9369	1.315	10001	10000

Nimble version

- Nimble can be set up with almost identical BUGS code

```
library(nimble)

GPCode<-nimbleCode({
for(i in 1:m){
mu[i]<-lambda[i]*expe[i]
y[i]~dpois(mu[i])
lambda[i]~dgamma(a,b)
}
a~dexp(10)
b~dexp(10)}
)
```

- File: Gamma_Poisson_Nimble_Congen90Data.txt

Set up

- Code
- Data
- Constants
- Inits

- Nimble requires that a separate list is specified for the data, and constants (any constant values used in the model).
- Initial values are also in a separate list

Lists

```
GPConsts<-
list(m=46,expe=c(1.0807,6.3775,0.622,6.6854,0.9142,1.0744,5.6
518,8.1682,0.5749,18.0989,2.174,1.6619,1.9321,1.6148,
1.6713,3.0819,1.7562,4.9952,0.9362,1.2001,6.1293,2.5604,15.8589,
2.9437,1.0399,7.276,0.9739,2.064,2.7206,2.8275,0.9425,
8.828,0.3644,1.775,1.5111,1.5111,2.5321,4.5836,3.9647,15.0264,0.73
2,10.8292,5.9848,1.4357,1.9949,6.9807))

GPData<-
list(y=c(0,7,1,5,1,1,5,16,0,17,4,0,0,1,1,7,1,3,0,0,8,2,13,7,0,8,0,3,2,4,
1,11,0,1,2,3,3,8,6,14,3,11,6,0,1,5))

GPInits<-list(a=10,b=10,lambda=rep(1.0,GPConsts$m))
```

nimbleModel

- nimbleModel can be used to produce a nimble model object for subsequent use

- E.g.

```
GP<-
nimbleModel(code=GPCode,name="GP",constants=GPConst
s,data=GPData,inits=GPInits)
```

- This can be used in nimbleMCMC or later in runMCMC

instead of directly specifying the inputs.

Model setup and running

- Two routes can be taken to run Nimble code
  - nimbleMCMC

  OR
    - Configure
    - Build
    - Compile
  - runMCMC

nimbleMCMC

- nimbleMCMC is a wrapper for the compiling and running of nimble code
- It automatically includes configure, compile, build, and running of the code specified
- It can run the original code , plus data, constants and inits OR can use a nimble object

nimbleMCMC default settings

nimbleMCMC(code, constants = list(), data = list(), inits, model, monitors, thin = 1, niter = 10000, nburnin = 0, nchains = 1, check = TRUE, setSeed = FALSE, progressBar = getNimbleOption("MCMCprogressBar"), samples = TRUE, samplesAsCodaMCMC = FALSE, summary = FALSE, WAIC = FALSE)

Simple use

```
mcmc.out<-
nimbleMCMC(code=GPCode,constants=GPConsts,
data=GPData,inits=GPInits,nchains=2,niter=10000,summary
=TRUE,WAIC=TRUE)
```

- mcmc.out now has results of the 10000 iteration run for 2 chains, no burnin, and has calculated the WAIC
- WAIC is a relative goodness of fit measure which can be used to compare models
- Smaller WAIC - > better model
- It is similar to DIC but more stable

nimbleModel and compileNimble

```
GP<-
nimbleModel(code=GPCode,name="GP",constants=GPC
onsts,data=GPData,inits=GPInits)

CGP<-compileNimble(GP)
```

- GP is a nimble model object
- CGP is a compiled model object
- Often we only need to use GP in running models

ConfigureMCMC

GPConf<-configureMCMC(GP,print=TRUE)

This sets up the samples used with the model GP

- You can add monitors here and enableWAIC also
- You can also suggest alternate samplers (such as slice samplers)
- print=TRUE will print the ordered list of samplers used.

Output

```
===== Monitors =====
thin = 1: a, b
===== Samplers =====
RW sampler (2)
  - a
  - b
conjugate sampler (46)
  - theta[]  (46 elements)
.
.
```

Adding monitors

- By default, Nimble reports the stochastic nodes
- In our example those are ‘a’, and ‘b’
- If you want other nodes reported you need to add monitors.

```
>GPConf$addMonitors(c("lambda"))
thin = 1: a, b, lambda
```

- This allows lambda to be monitored also

Build and compile and run

```
>GPMCMC<-buildMCMC(GPConf)
>CGPMCMC<-compileNimble(GPMCMC,project=GP)
```

- Once compiled CGPMCMC can be run

```
>niter<-10000
>set.seed(1)
>samples<-runMCMC(CGPMCMC,niter=niter)
```

- This runs one chain for 10000 iterations with no burnin

runMCMC default settings

>runMCMC(mcmc, niter = 10000, nburnin = 0, thin, thin2, nchains = 1, inits,

setSeed = FALSE, progressBar = getNimbleOption("MCMCprogressBar"),

samples = TRUE, samplesAsCodaMCMC = FALSE, summary = FALSE,WAIC = FALSE)

- runMCMC normally returns a matrix with columns corresponding to monitored nodes
- If you set samplesAsCodaMCMC=TRUE then a CODA formatted object results and can be processed using the CODA R package

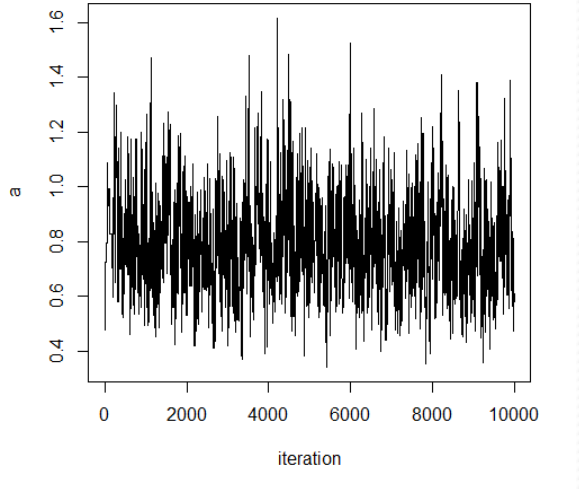
runMCMC example

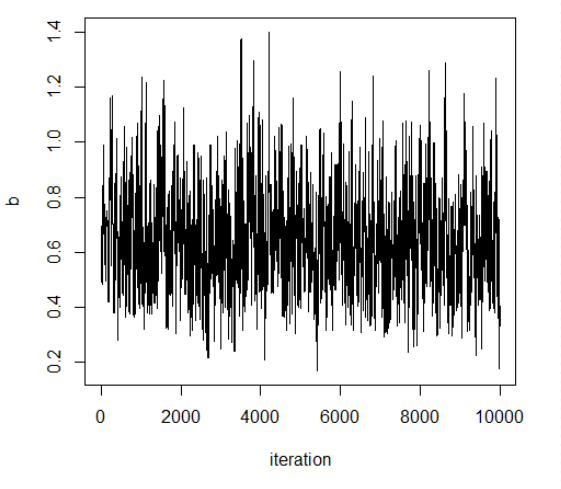
```
> niter<-10000
> set.seed(1)
> samples<-runMCMC(CGPMCMC,niter=niter)
running chain 1...
|------------|------------|------------|------------|
|-----------------------------------------------------|
> head(samples)
```

	a	b	lambda[1]	lambda[2]	lambda[3]	lambda[4]	lambda[5]
[1,]	0.475074	0.49102	0.878816	0.509698	1.361868	0.94172	1.157481
[2,]	0.723487	0.556308	0.419284	0.996098	0.194962	0.767394	0.7458
[3,]	0.723487	0.556308	0.072694	0.731913	0.985618	0.74309	0.606311
[4,]	0.723487	0.556308	1.051699	1.364117	3.014887	0.610594	1.28902
[5,]	0.723487	0.556308	0.144022	0.884983	0.809499	0.58816	4.017854

Trace Plots

```
>plot(samples[,"a"],type="l",xlab="iteration",ylab=expression(a))
>plot(samples[,"b"],type="l",xlab="iteration",ylab=expression(b))
```





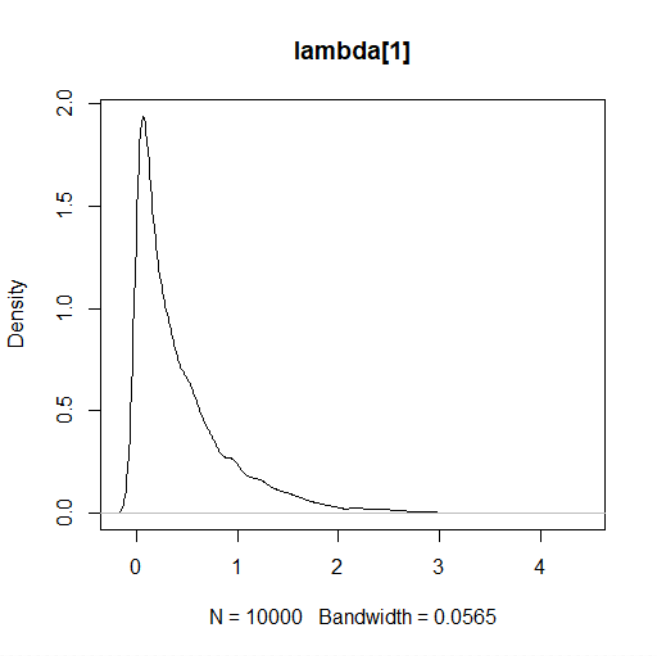
Convert to dataframe

- If desired you can convert to a dataframe

>samplesDF<-
data.frame(samples)

>plot(density(samplesDF$th
eta.1.),main="theta[1]")

- This gives a density plot of the theta[1] sample



Summary

- BUGS code is assigned via nimbleCode
- A separate list for data, constants, inits
- A quick run using nimbleMCMC
- For maximum control use build, configure, compile and runMCMC
- Output is in the form of matrix of sampled parameters
- Can be converted to a dataframe
- CODA can be used for convergence diagnostics

Notes on Convergence

- The R package CODA provides a range of diagnostics for convergence, e.g.
  - Geweke diagnostic (single chain)
  - Brooks Gelman Rubin (BGR) (multiple chains)
  - Also time series plots available (acfplot)

- Commands:
  - Gelman.diag
  - Geweke.diag

```
>samples<-
runMCMC(CGPMCMC,niter=niter,samplesAsCodaMCMC=T
RUE)
```

Library CODA and smfsb

- Assuming you stored the run in samples (as a coda format)
- For single chain use

```
>geweke.diag(head(samples),frac1=0.4,frac2=0.5)
```

- Will give a diagnostic for each parameters
  - If it has converged then it will be within +-2

- For multiple chains (i.e. nchains>=2)

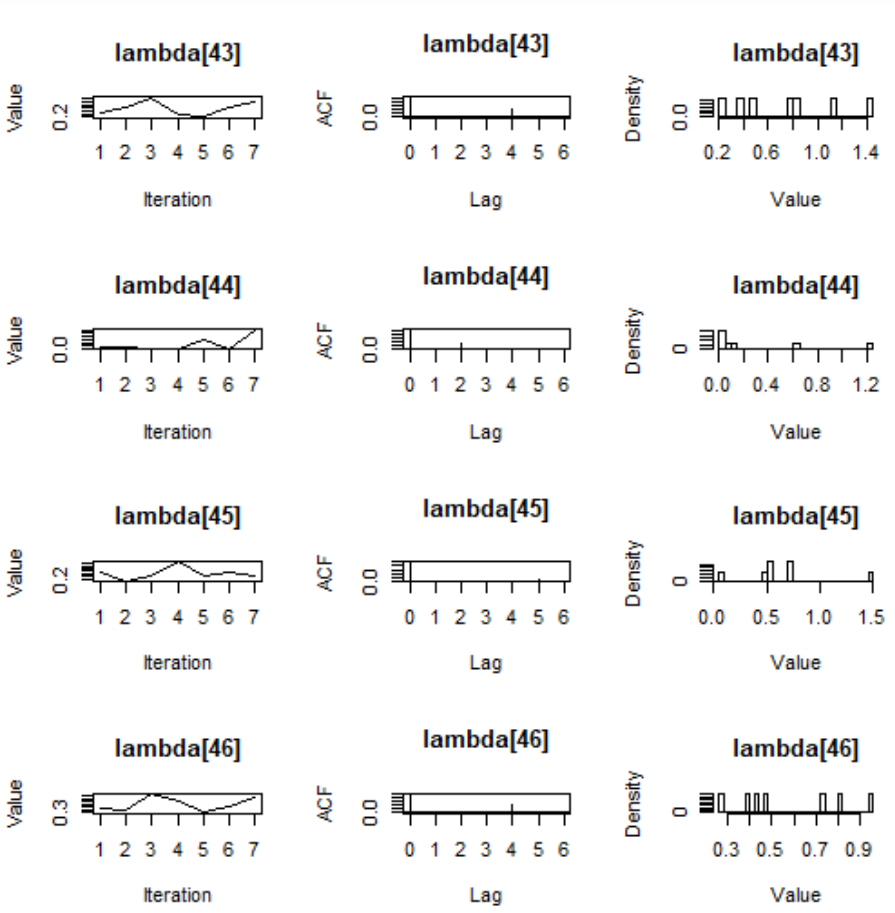
```
>gelman.diag(head(samples))
```

```
>library(smfsb)
>mcmcSummary(head(samples),show=FALSE)
```

- This gives multiple plots for each parameter

smfsb output example

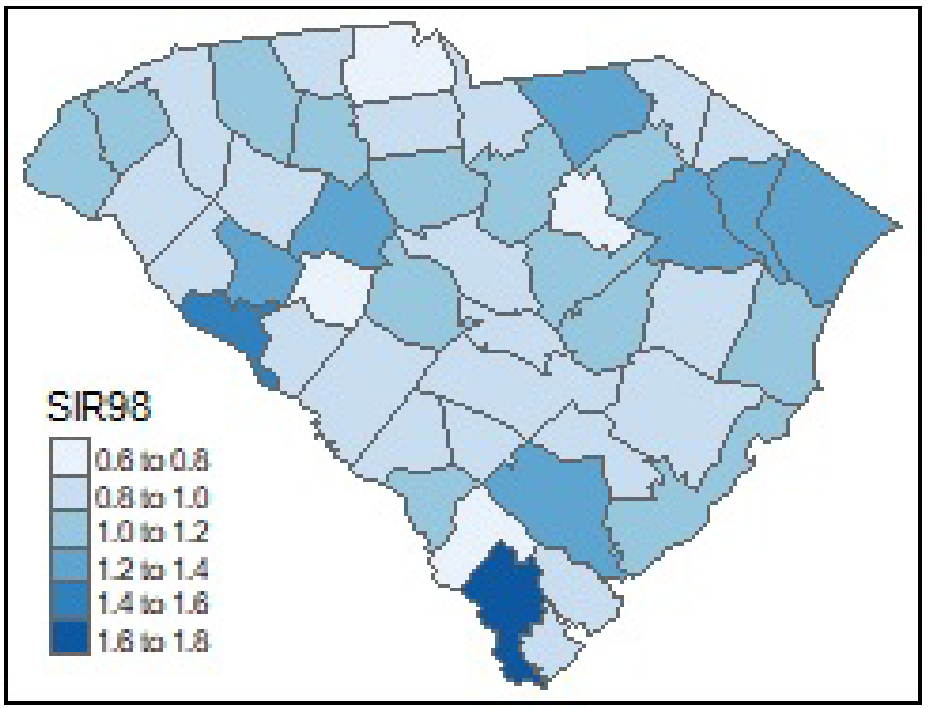
- This library gives trace plot, acf plot and density for each parameter



Extending the PG model

- The Poisson–Gamma model is limited as it cant be easily extended to include covariates or confounding effects.
- Because of this, researchers tend to use more flexible models whereby the risk parameters are linked to a linear or non-linear predictor.
- For the Poisson data model this is a log link and leads to a log normal model
- For the finite population case: the binomial model has a logit link (usually) and this leads to a logistic normal model

Data example: SC respiratory cancers 1998



Data

asd<-list(m=46,Y1998=c(18,90,10,120,12,14,76,96,10,256,37,23,40,29,36,
55,21,63,15,19,129,47,260,60,10,184,22,45,43,44,10,171,11,
34,22,34,51,63,90,201,10,202,87,25,25,91),
Exp98=c(19.334642001146, 105.221991510865, 8.9954123633133, 126.211287025262,
12.9499400671852, 17.0850039703209, 85.5262771111914, 107.178846922884,
11.0291918950188, 248.419380066852, 38.5954996425929, 27.0027208298727,
32.2453350684913, 24.1871410613557, 29.3284980403873, 52.0933278275436,
23.3496100847714, 69.1791167378613, 15.7011547559647, 17.5779462883105,
98.0421453601469, 42.1724712080047, 277.747093167242, 49.9402374163248,
15.0708479385354, 137.177683720537, 13.3400552455942, 38.1425892644401,
46.222761591486, 49.646669857522, 16.011990994697, 161.116783742905,
7.49225226944375, 27.1667732892036, 23.2255895652772, 27.0506021696774,
50.282471254929, 68.9687528187193, 84.0568694371842, 241.020535657027,
13.3636034454982, 194.239681727817, 84.0882670370562, 23.9367452023769,
29.1377576211652, 121.126445726))

Log/logistic normal GLMMs

- Often discrete spatial models can be set up in the form of log normal generalized linear mixed models (GLMMs) when Poisson data is found

$$y_i \sim Pois(e_i \theta_i)$$
$$log(\theta_i) = ...$$

- For binomial data then a logistic GLMM arises

$$y_i \sim bin(p_i, n_i)$$
$$logit(p_i) = ...$$

LMM or GLMM

$$y_i = x_i^T \beta + z_i^T \gamma + e_i$$

or

$$E(y_i) = \mu_i;\ f(y) \text{ non-Gaussian (continuous/discrete)}$$

$$g(\mu_i) = x_i^T \beta + z_i^T \gamma$$

where x is a design matrix of covariates

and z is a design matrix of random effects

and γ is a unit vector

Some examples of spatial mixed models

- Geostatistical data:

y_i : outcome measured at location s_i

$s_i = \{s_1, s_2\}$

$\boldsymbol{y} : N(\boldsymbol{\mu}, \Sigma)$

$\Sigma_{ij} = cov(s_i, s_j)$

e.g.

$y_i = x_i^T \beta + e_i;\ \boldsymbol{e} \sim N(0, \Sigma)$

Examples

- Small area health data:

$$y_i \sim Pois(\mu_i)$$

$$\mu_i = e_i \theta_i$$

$$\log(\mu_i) = \log(e_i) + \log(\theta_i)$$

$$\log(\theta_i) = x_i^T \beta + z_i^T \gamma$$

Special Cases

1) simple trend: $\log(\theta_i) = \beta_0 + \beta_1 s_{1i} + \beta_2 s_{2i}$

where (s_1, s_2) are centroid coordinates

2) random intercept: $\log(\theta_i) = \beta_0 + z_{1i}$

$z_{1i} \sim N(0, \tau_1^{-1})$; $\beta_0 \sim N(0, \tau_0^{-1})$

3) convolution model: $\log(\theta_i) = \beta_0 + z_{1i} + z_{2i}$

$z_{2i} \mid \boldsymbol{z}_{2j \neq i} \sim N(\bar{z}_{\delta_i}, \tau_1^{-1} / n_{\delta_i})$

Uncorrelated heterogeneity (UH)

- UH model (random intercept)
  - Uncorrelated Noise model
  - Baseline risk model
  - Assumes no spatial correlation or trend
  - Zero mean Gaussian prior distributions for effects
    - Intercept and random effect

$$z_{1i} \sim N(0, \tau_1^{-1}); \; \beta_0 \sim N(0, \tau_0^{-1})$$

Precisions

- Gaussian distributions are governed by precisions
- Precision = 1/variance
- What about a prior distribution for the precision?
- This is known as a *hyper-prior* distribution
- Proposals:

$Gamma$:

$\tau \sim Ga(a,b)$

a,b usually <1 for non-informativeness

$SD-uniform$:

$\tau^{-1/2} \sim U(0,c)$

- SD-uniform used frequently especially on WinBUGS

A note on prior distributions

- All parameters in Bayesian models must have prior distributions (or they are FIXED i.e. known values)
- If a parameter is fixed then it is **not** estimable
- If you want to estimate a parameter in a Bayesian model then it MUST have a prior distribution
- This is very important for precisions (especially for random effects)

$$z_{1i} \sim N(0, \tau_1^{-1}); \ \beta_0 \sim N(0, \tau_0^{-1})$$

UH log normal model

- We will use the SC respiratory cancers 1998 data for this example
- This is a random effect (random intercept) model
- It picks up unstructured noise
- Nimble code file:

lognormal_nimble_model _and_code.txt

Code

```
library(nimble)
LNmodel<-nimbleCode({
   for(i in 1:m){
   Y1998[i]~dpois(mu[i])
   mu[i]<-Exp98[i]*theta[i]
   log(theta[i])<-a0+z1[i]
z1[i]~dnorm(0,tauZ)
LDev[i]<--2*(Y1998[i]*log(mu[i]+0.001)-(mu[i]+0.001)-lfactorial(Y1998[i]))
   }
Dev<-sum(LDev[1:m])
 a0~dnorm(0,tau0)
tauZ~dgamma(2,0.5)
tau0~dgamma(2,0.5)
   })
```

Correlated Heterogeneity (CH)

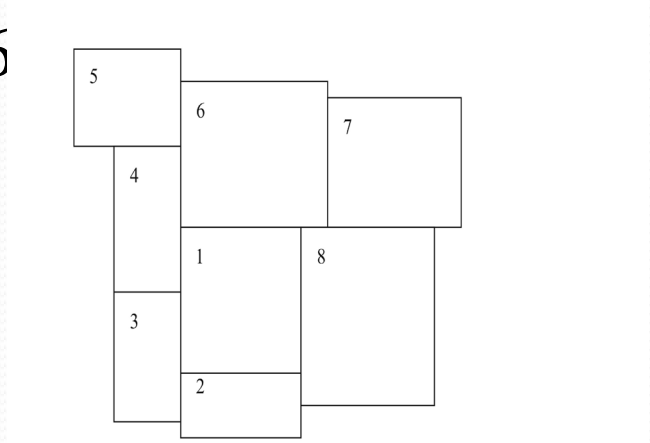
- This component is where correlation is introduced.
- Some times called *clustering* or *spatially-structured*
- We could assume a variety of forms for spatial correlation
- We often use a CAR model as it is simple and easy to fit
- It can be an improper or proper prior distribution
- The ICAR (improper) is easy to fit on Nimble and WinBUGS
- The ICAR does not have a correlation parameter: the precision controls the variation and correlation
- It is adaptive in that the variation depends on the neighborhoods

$$u_i \mid u_{j \neq i} \sim N(\bar{u}_{\delta_i}, \tau_1^{-1} / n_{\delta_i})$$

$$\bar{u}_{\delta_i} = \sum_{k \in \delta_i} u_k / n_{\delta_i}$$

Dependencies

- Often adjacencies are used as proxies for dependence
- adjacency: 2,3 ,4 ,6
- hence



$$\bar{u}_{\delta_i} = \sum_{j \in \delta_i} u_j / n_{\delta_i}$$

$$\bar{u}_{\delta_1} = (u_2 + u_3 + u_4 + u_6 + u_8)/5$$

ICAR model on nimble

```
 CONVmodel<-nimbleCode({
    for(i in 1:m){
    Y1998[i]~dpois(mu[i])
    mu[i]<-Exp98[i]*theta[i]
    log(theta[i])<-a0+u[i]
LPoisY[i]<-Y1998[i]*log(mu[i]+0.001)-(mu[i]+0.001)-lfactorial(Y1998[i])
PoisY[i]<-exp(LPoisY[i])
LDev[i]<--2*LPoisY[i]
    }
u[1:m]~dcar_normal(adj[1:L],wei[1:L],num[1:m],tauU,zero_mean=1)
for(k in 1:L) {wei[k] <- 1 }
Dev<-sum(LDev[1:m])
 a0~dnorm(0,tau0)
tau0~dgamma(2,0.5)
tauU~dgamma(2,0.5)
    })
```

Spatial ingredients

- Adjacency vector (adj[])
- Number of neighbors vector (num[])

- These specify the spatial structure of neighbors for the ICAR prior distribution

Typical Poisson convolution model

- Describing spatial variation and confounding using random effects: a BYM or convolution model

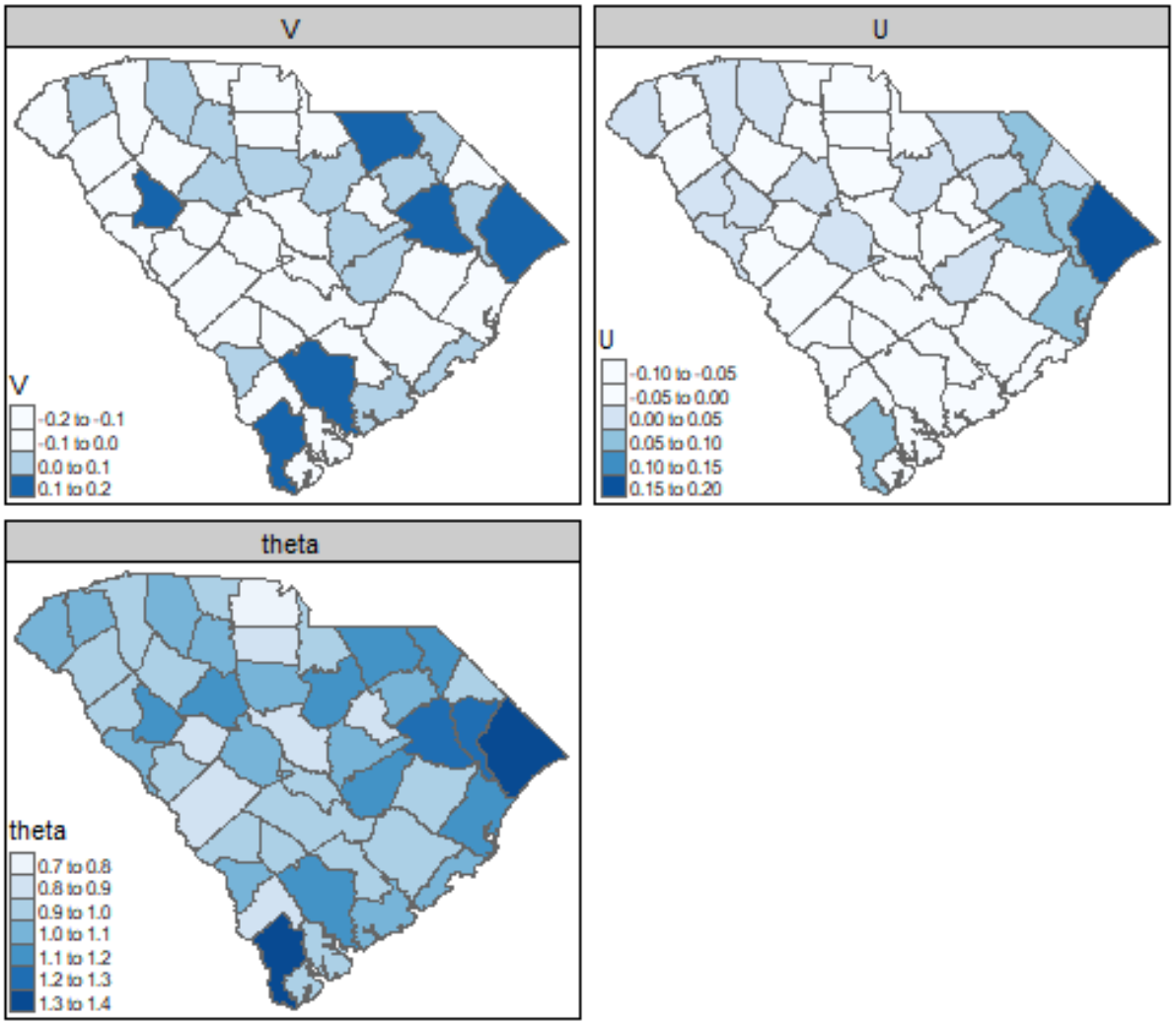
$$y_i \sim Pois(e_i \theta_i)$$
$$log(\theta_i) = \alpha_0 + v_i + u_i$$
$$\alpha_0 \sim N(0, \tau_0^{-1})$$
$$v_i \sim N(0, \tau_v^{-1})$$
$$u_i \sim ICAR(\tau_u^{-1})$$
$$\tau_* \sim Ga(2.0, 0.5)$$

Nimble convolution model

- File: CONV_nimble_model_and_code.txt

```
CONVmodel<-nimbleCode({
   for(i in 1:m){
   Y1998[i]~dpois(mu[i])
   mu[i]<-Exp98[i]*theta[i]
   log(theta[i])<-a0+v[i]+u[i]
prex[i]<-step(theta[i]-1)
v[i]~dnorm(0,tauV)
LDev[i]<--2*(Y1998[i]*log(mu[i]+0.001)-(mu[i]+0.001)-lfactorial(Y1998[i]))
   }
u[1:m]~dcar_normal(adj[1:L],wei[1:L],num[1:m],tauU,zero_mean=1)
for(k in 1:L) {wei[k] <- 1 }
Dev<-sum(LDev[1:m])
 a0~dnorm(0,tau0)
tauV~dgamma(2,0.5)
tau0~dgamma(2,0.5)
tauU~dgamma(2,0.5)
   })
```

Convolution model: Posterior mean maps of the random effects and relative risk



Other results

- Mean deviance: 295.45
- DIC: 325.69
- WAIC: 317.43 (pWAIC: 16.29)

- Posterior estimate of intercept (a0) :

Mean	St.Dev.	95%CI_low	95%CI_upp
0.01359462	0.04221736	-0.06570649	0.09851966

What about fixed effects ?

- Covariates can be added easily in the code
- For example:
  - Both poverty and median income appear in the data.frame so we can add these to the model

log(theta[i])<-a0+a1*pov[i]+a2*inc[i]+v[i]+B[i]

Must also assume prior distributions for a1 and a2

Nimble versus CARBayes

- Nimble: provides a extensive language for modeling
  - Allows nested indexing so that truly multi-level models are possible
  - Regression models are straightforward to fit
  - Nimble is much faster than WinBUGS and can handle large datasets quickly
- CARBayes: can fit spatial models quickly
  - It is useful for regression modeling (flexible links)
  - Reports only median results BUT uses MALA
  - Only handles outcome missingness

Nimble v INLA

- INLA is another R package which can fit Bayesian DM models
- It is based on a numerical integral approximation as opposed to McMC.
- It can fit models very efficiently
- It provides a user interface similar to CARBayes, fitting models via commands like LM function in R
- Very useful for quick model fitting with simpler spatial regression models
- Doesn't have flexibility of nimble programming

Nimble v STAN

- STAN allows the fitting of CAR models and has been a rival to BUGS as a programming language.
- The difference in set up of models is major although the R package `brms` is user friendly.
- STAN uses fast HMC for sampling, but nimble is also implementing this in a new version and so will also have access to these algorithms.
- There are limitations on discrete prior distributions in STAN
- Otherwise it is a good system to use.

References

1. **Lawson, A. B.** (2020) Nimble for Bayesian Disease Mapping. *Spatial and Spatio-temporal Epidemiology*, https://doi.org/10.1016/j.sste.2020.100323
2. **Lawson, A. B.** (2018) *Bayesian Disease Mapping: hierarchical modeling in spatial epidemiology* CRC Press , New York 3rd Ed
3. **Lawson, A.** B. (2021) Using R for Bayesian Spatial and Spatio-temporal Health data Modeling, CRC Press.

ARTICLE IN PRESS
JID: SSTE [m5G;February 25, 2020;13:33]
Spatial and Spatio-temporal Epidemiology xxx (xxxx) xxx

Contents lists available at ScienceDirect

Spatial and Spatio-temporal Epidemiology

journal homepage: www.elsevier.com/locate/sste

NIMBLE for Bayesian Disease Mapping

Andrew B Lawson

Department of Public Health sciences, Medical University of South Carolina, Charleston, SC, USA

ARTICLE INFO

Article history:
Received 11 December 2019
Revised 23 January 2020
Accepted 23 January 2020
Available online xxx

Keywords:
Nimble
BHM
BDM
Bayesian
Gamma poisson
Log normal
BYM
convolution

ABSTRACT

This tutorial describes the basic implementation of Bayesian hierarchical models for spatial health data using the R package nimble. To quote the nimble R description:

```
A system for writing hierarchical statistical models largely compatible
with 'BUGS' and 'JAGS', writing nimbleFunctions to operate models and
do basic R-style math, and compiling both models and nimbleFunctions via
custom-generated C++. 'NIMBLE' includes default methods for MCMC, particle
filtering, Monte Carlo Expectation Maximization, and some other tools. The
nimbleFunction system makes it easy to do things like implement new MCMC
samplers from R, customize the assignment of samplers to different parts
of a model from R, and compile the new samplers automatically via C++
alongside the samplers 'NIMBLE' provides.
```

Examples of the use of the package for a small range of Bayesian Disease Mapping (BDM) models is explored and focus on different approaches to model fitting and analysis are discussed. Examples of publicly available small area health data is used throughout.

1. Introduction

In its simplest form nimble can take BUGS or JAGS code for hierarchical models (BHMs), parse and interpret the code to translate the models into posterior samplers and implement them in C++. As the samplers are written in a language directly, they can often run considerably faster than the original BUGS or JAGS code. In addition, nimble allows the extension of the original languages by providing facilities for the addition of distributions and calling compiled code and R functions. In this tutorial a dataset of county level respiratory cancer counts for the year of 1998 for the US state of South Carolina is examined for use with spatial models. Expected counts in counties are also computed from the standard statewide rate. Fig. 1 displays the standardized incidence ratio for the 1998 data in South Carolina counties.

In addition to spatial models, code for a typical spatio-temporal model is showcased as is multivariate model code. An extend variety of examples, including multi-scale, multivariate and other latent structure models are available to download with embedded data from the site: https://github.com/Andrew9Lawson/Bayesian_DM_Nimble_code

2. Bayesian modeling and computation

A brief introduction to Bayesian modeling is provided here. For considerably more detail please refer to Lawson (2018), and references therein to more general texts. It is assumed that the reader has some acquaintance with Bayesian methods and that the focus is simply on a software development that can help the modeling process. A basic model setup consist of a data model, or distribution, for an outcome: $f(y|\boldsymbol{\theta})$ which for a sample of y values can form a likelihood $L(\mathbf{y}|\boldsymbol{\theta})$, and prior distributions for the parameters in $\boldsymbol{\theta}$: $\Pr(\boldsymbol{\theta})$. A posterior distribution is formed by their product $\Pr(\boldsymbol{\theta}|\mathbf{y}) \propto L(\mathbf{y}|\boldsymbol{\theta}).Pr(\boldsymbol{\theta})$. It is this posterior distribution that is used to make inference about the parameters $\boldsymbol{\theta}$. Posterior sampling is a general stochastic approach to deriving this inference, whereby an iterative algorithm is used to simulate from the posterior. Markov chain Monte Carlo (MCMC) is usually used for this purpose. MCMC involves simulating sample parameter values iteratively until a converged state is reached where the values correspond to a sample from the posterior distribution. For simple models convergence could take place in a few hundred iterations but for complex models with a large number of parameters, convergence may require many thousands of steps. The nimble package provides facilities for running MCMC chains of parameters with fixed number of iterations. It is then important to assess whether convergence has

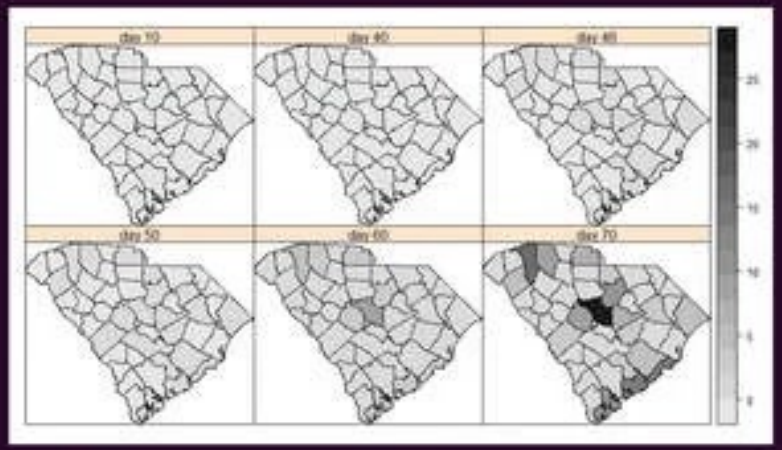
E-mail address: lawsonab@musc.edu

https://doi.org/10.1016/j.sste.2020.100323
1877-5845/

Please cite this article as: A.B. Lawson, NIMBLE for Bayesian Disease Mapping, Spatial and Spatio-temporal Epidemiology, https://doi.org/10.1016/j.sste.2020.100323

The R Series

Using R for Bayesian Spatial and Spatio-Temporal Health Modeling



Andrew B. Lawson

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Resources

- My home page has various code :

http://people.musc.edu/~abl6/

- GitHUB repository for various code examples (Nimble, Winbugs, INLA, CARBayes):

https://github.com/Andrew9Lawson

Thank You!

- Contact : lawsonab@musc.edu