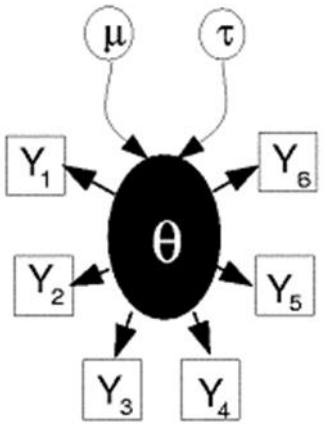
Introduction to the BUGS language



BUGS

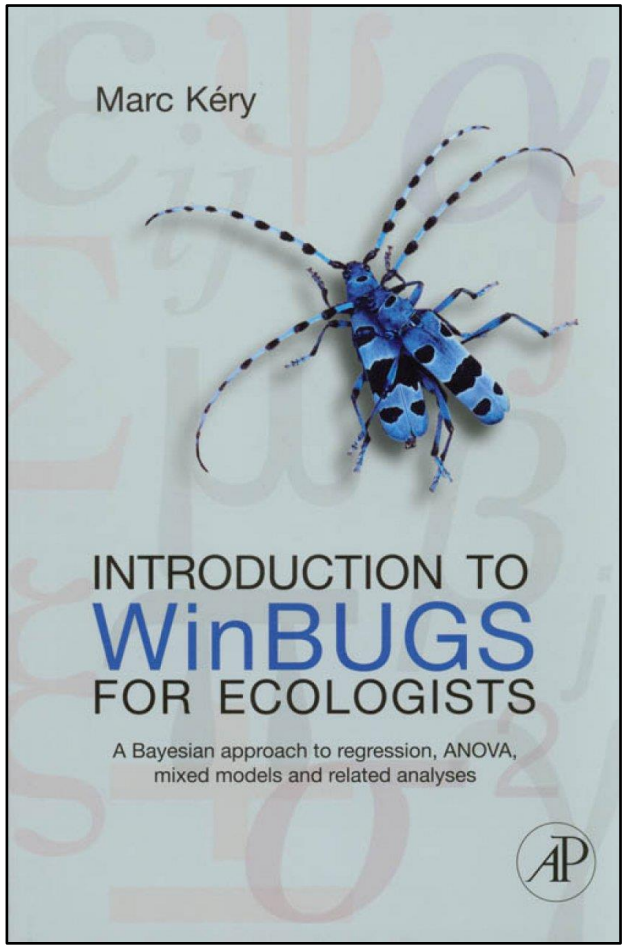
Spatial Capture-Recapture Workshop,

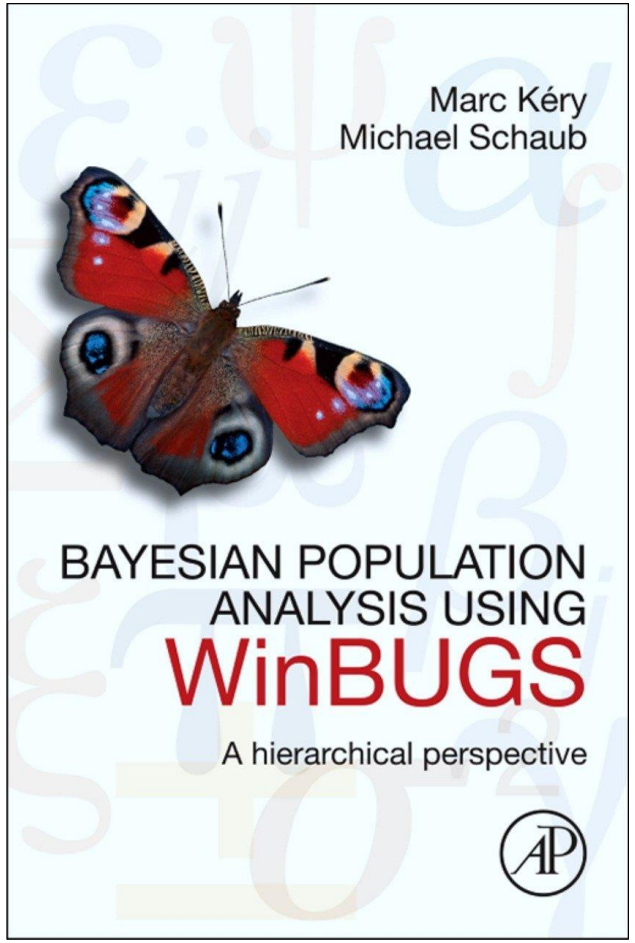
Cornell University, Ithaca, NY

April 2014

Thanks to…

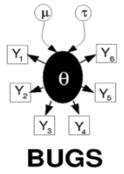
- Marc Kéry, who let us use his workshop materials…





Overview

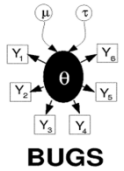
- Bayesian analysis by MCMC (***Refresher***)
- What is BUGS?



- Using WinBUGS as a stand-alone application (***Demo***)

- Using BUGS through R (***Demo***)

  `WinBUGS → R2WinBUGS | JAGS → rjags or R2JAGS`

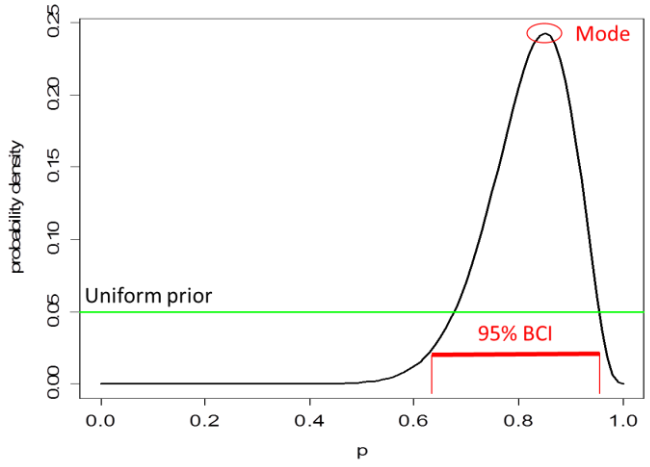


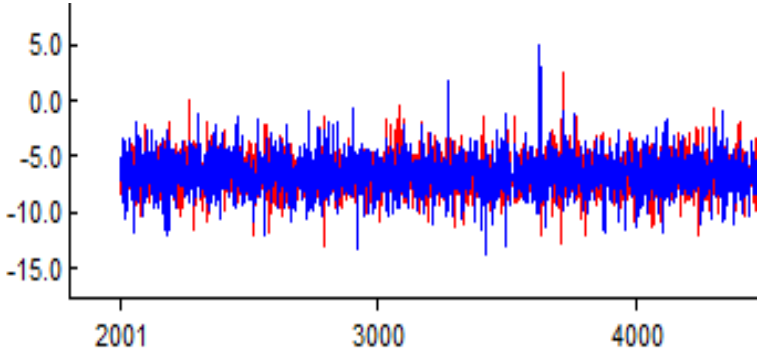
→

- Using BUGS through R (***Exercises***)
  - Poisson & Binomial regression

- Random effects model in BUGS (***Exercises***)
  - Binomial regression with random effects
  - Occupancy model for estimating number of occupied patches

Bayesian Inference and Markov chain Monte Carlo (MCMC) Sampling





$$[\theta|y] = \frac{[y|\theta][\theta]}{[y]}$$

A [very!] quick refresher

The Bayesian paradigm

- **Data**
  - observed realization of a random process
    - y

- **Parameters**
  - fixed & unknown data generating value, realization of a random process
    - θ

- **Uncertainty about parameters**
  - Based on variation in data and evaluated using *posterior distribution*

Bayes‘ Rule

$[\theta\ |y]$ Posterior distribution of θ given the data y

$[y|\theta]$ Likelihood of the data y given the model [parameters θ]

$[\theta]$ Prior probability distribution of parameters θ

$[y]$ Marginal distribution of data y

$$[\theta|y] = \frac{[y|\theta][\theta]}{[y]}$$

Bayes‘ Rule

$[\theta\ |y]$ Posterior distribution of θ given the data y

$[y|\theta]$ Likelihood of the data y given the model [parameters θ]

$[\theta]$ Prior probability distribution of parameters θ

$[y]$ Marginal distribution of data y

$$[\theta|y] = \frac{[y|\theta][\theta]}{[y]}$$

$[y]$: In practice, hard, even impossible, to compute

Recent method developments that do not require mathematical understanding

$$[\theta|y] \propto [y|\theta][\theta]$$

Bayes‘ Rule

$[\theta\ |y]$ Posterior distribution of θ given the data y

$[y|\theta]$ Likelihood of the data y given the model [parameters θ]

$[\theta]$ Prior probability distribution of parameters θ

$$[\theta|y] \propto [y|\theta][\theta]$$

Prior probability distribution $[\theta]$

- The probability distribution reflecting our prior knowledge about an unknown parameter/quantity *before* any evidence is taken into account, i.e. *prior*

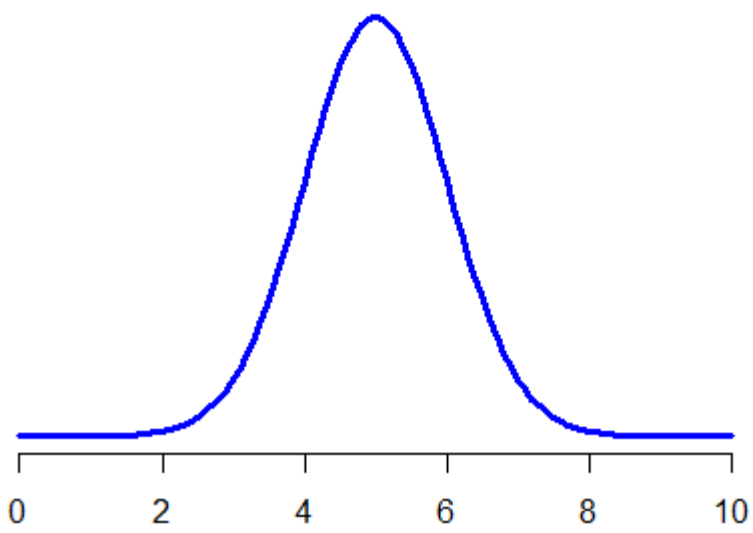
Prior probability distribution $[\theta]$

- The probability distribution reflecting our prior knowledge about an unknown parameter/quantity *before* any evidence is taken into account, i.e. *prior*

Informative priors

- Way of incorporating prior knowledge about parameters into the analysis

- Will influence parameter estimate, the magnitude of the influence depends on relative 'strength' of $[y|\theta]$ vs. $[\theta]$

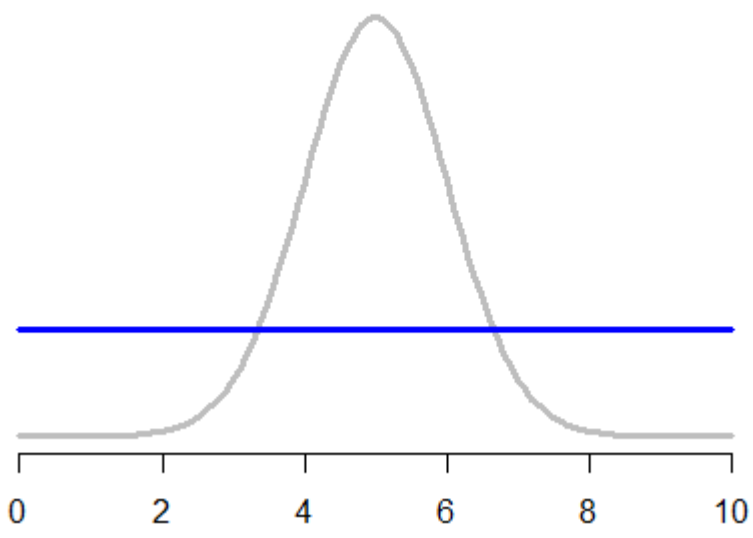


Prior probability distribution $[\theta]$

- The probability distribution reflecting our prior knowledge about an unknown parameter/quantity *before* any evidence is taken into account, i.e. *prior*

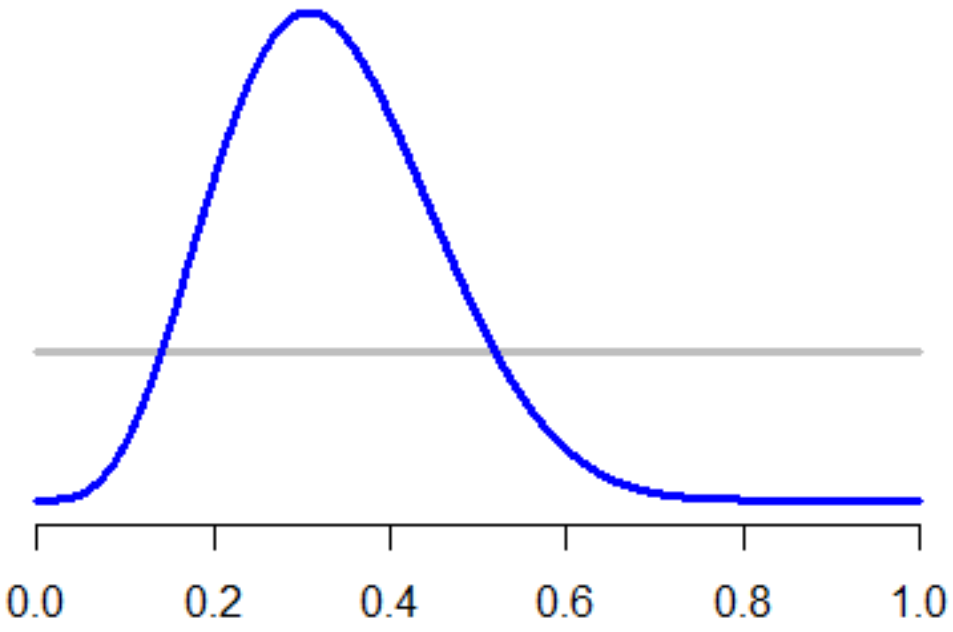
Uninformative/vague priors

- Express lack of prior knowledge about parameters
- Generally preferred for an objective analysis
- Most widely applied

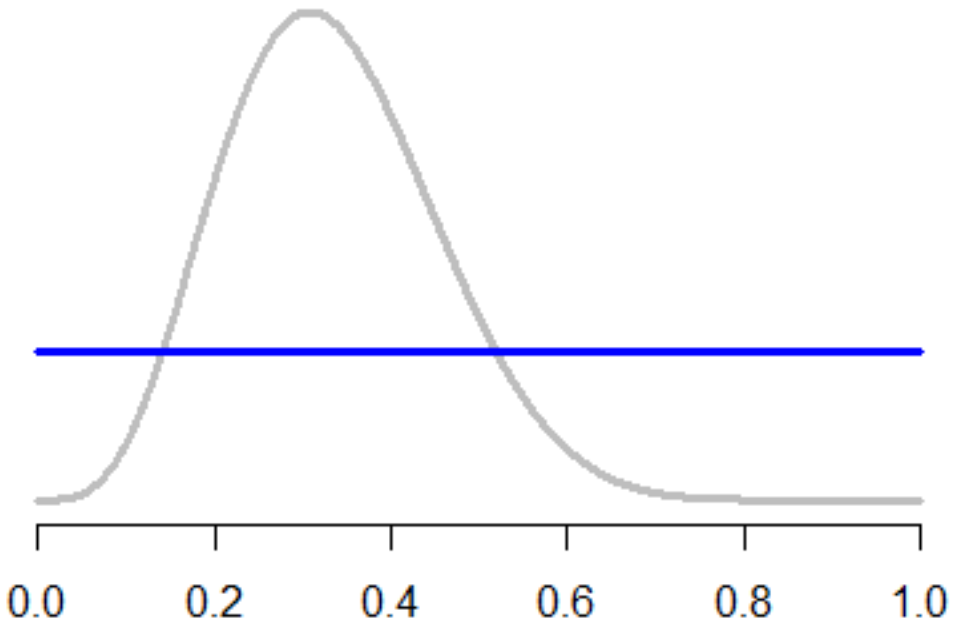


Prior probability distribution – detection probability $[p]$

Informative prior
Beta(5,10)



Uninformative/flat prior
Beta(1,1)



Likelihood of the data $[y|\theta]$

Say we go out and survey $N = 30$ ponds and find frogs present in $y = 12$ ponds (and we assume y is observed perfectly!!). Then the binomial model and likelihood is:

$$y \sim Binomial(n, \theta)$$

$$[y|\theta] = \theta^{y}(1-\theta)^{n-y}$$

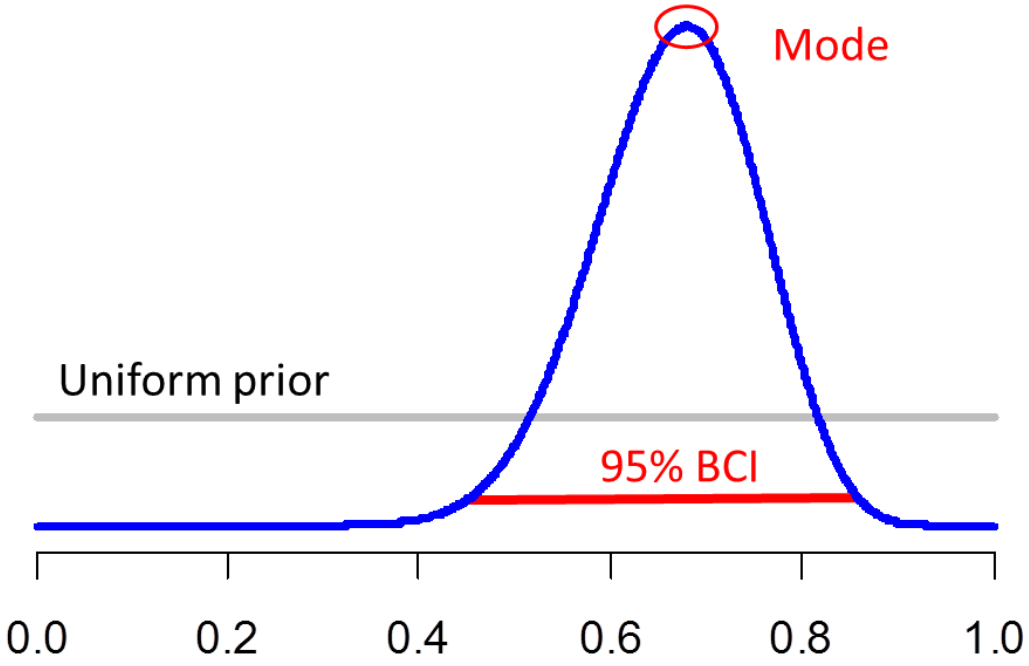
$$[12|\theta] = \theta^{12}(1-\theta)^{18}$$

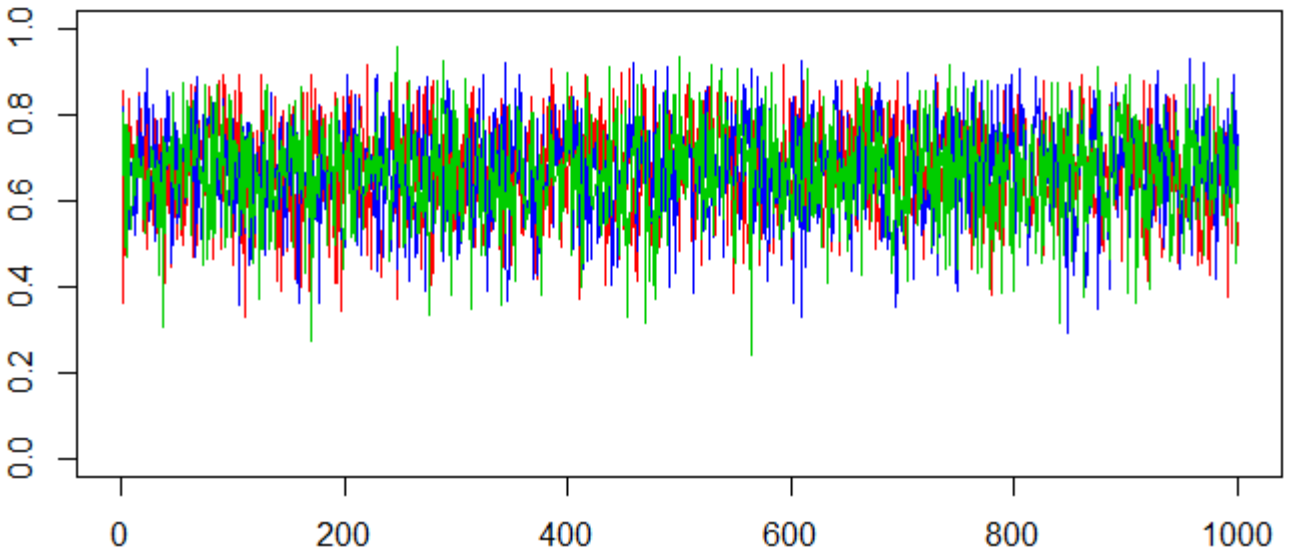
So, we can plug in different values of θ and calculate the probability of checking 30 ponds and finding 12 with frogs.

Posterior inference $[\theta|y]$

- Inference does not focus on estimating a single point value but on characterizing an entire distribution $[\theta|y]$

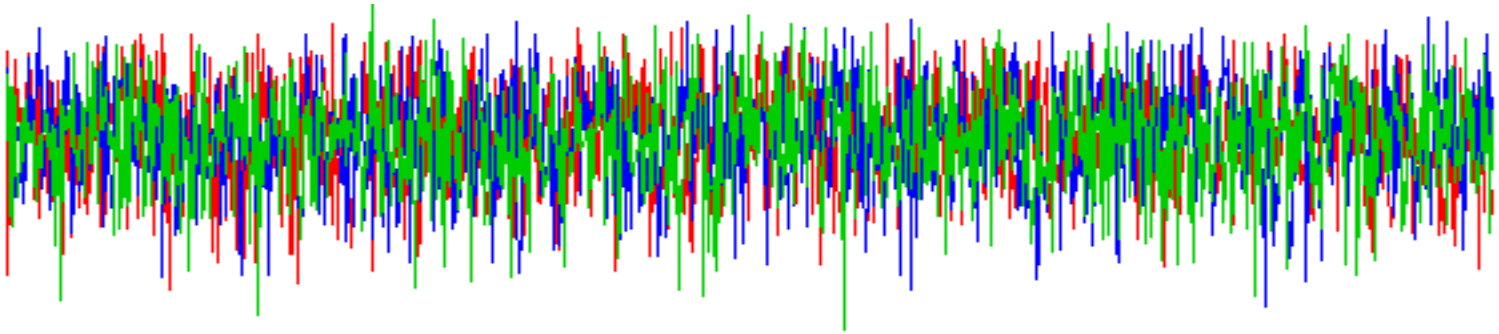
- Posterior can be characterized by Markov chain Monte Carlo (**MCMC**) simluation





Markov chain Monte Carlo - MCMC

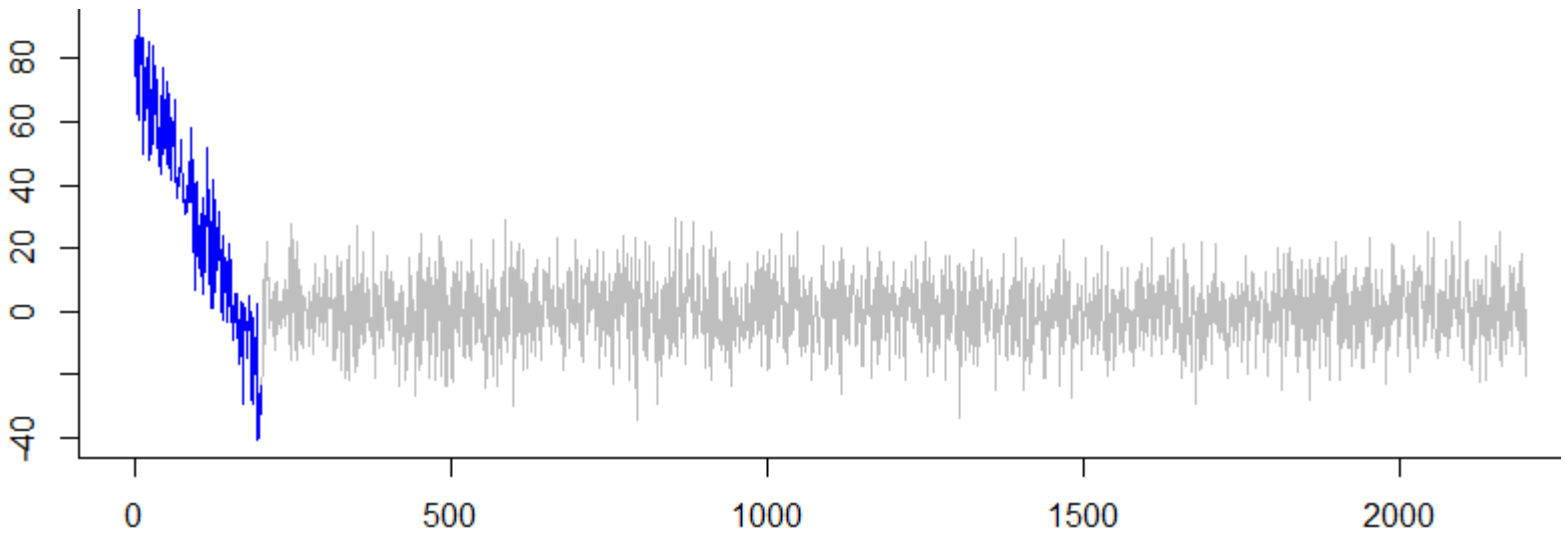
- Simulation-based evaluation of posterior distribution
- Generates random samples from the posterior
- Markov property: the value of each iteration idepends on the preceding value
- MCMC algorithms are constructed so that the Markov chain converges to the posterior distribution



MCMC – terminology

“Burn-in”

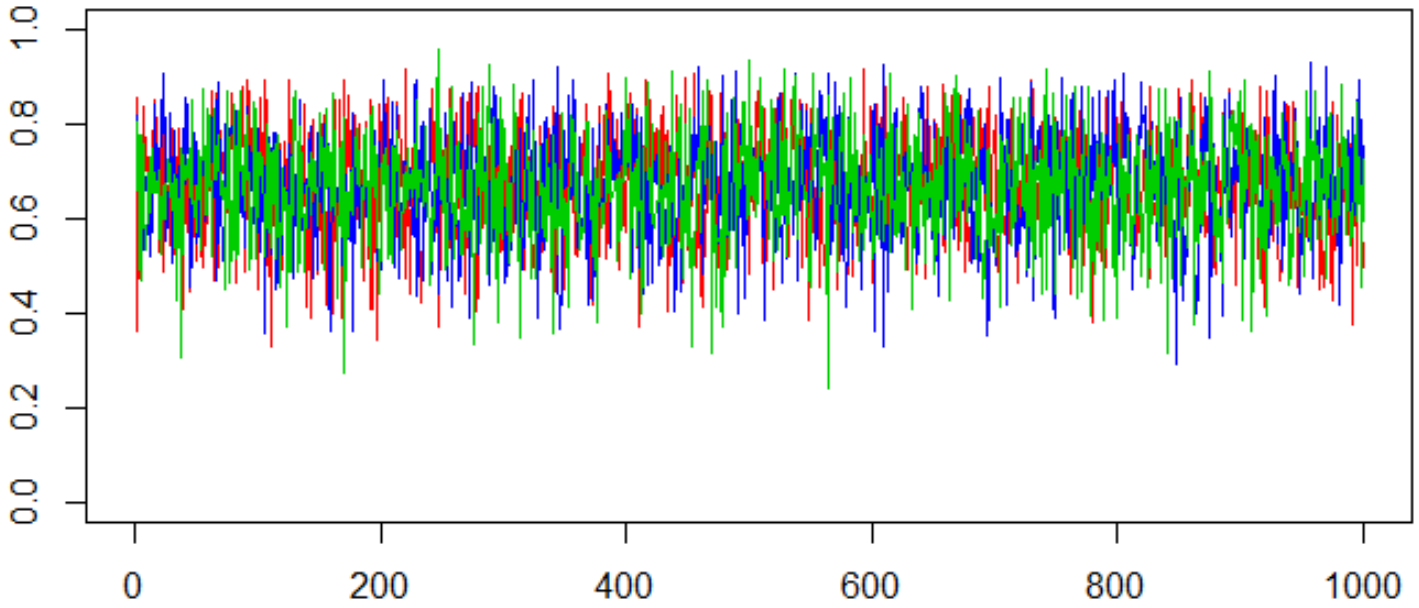
- Chains are started at “*initial values*”
- First set of iterations are influenced by initial values
- Transition phase from initial values to stationary distribution = **burn-in**
- Needs to be discarded!!



MCMC – terminology

"Convergence"

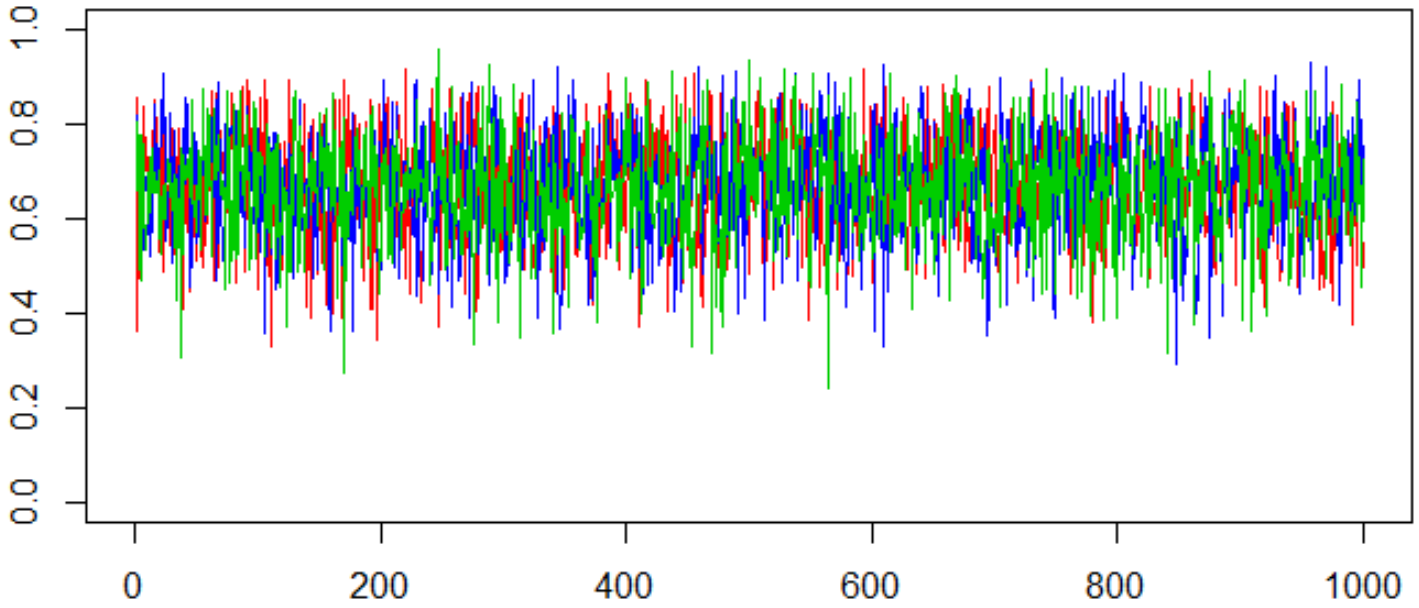
- Has the chain reached its "*stationary distribution*" (i.e. the posterior)?
- We can assess convergence using:
  - ➢ Trace plots – we want to see "grassy" chains!



MCMC – terminology

"Convergence"

- Has the chain reached its "*stationary distribution*" (i.e. the posterior)?
- We can assess convergence using:
  - ➢ Gelman-Rubin-statistic ($\hat{R}$); compares within- and between-chain variation (**<1.1**)



MCMC – terminology

“Posterior sample size”; “Monte Carlo error”

- MCMC samples are not independent because of the Markov property
- Effective sample size is the number of samples after accounting for *autocorrelation*
- Determines the MC error – “noise“ introduced into the sample by the stochastic MCMC process
- You can reduce the MC error by running longer chains
- Rule of thumb is that MC error should be <5% (1%)

References

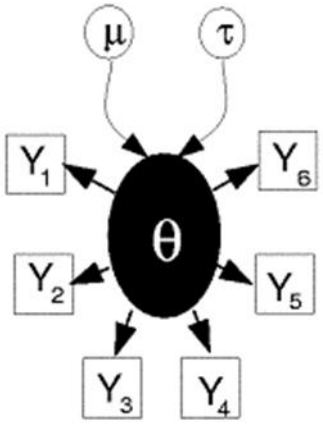
- Lunn, D. J., Best, N., & Whittaker, J. C. (2009). Generic reversible jump MCMC using graphical models. *Statistics and Computing*, *19*(4), 395-408.

- Lunn, D. J., Whittaker, J. C., & Best, N. (2006). A Bayesian toolkit for genetic association studies. *Genetic epidemiology*, *30*(3), 231-247.

- Lunn, D., Spiegelhalter, D., Thomas, A., & Best, N. (2009). The BUGS project: Evolution, critique and future directions. *Statistics in medicine*, *28*(25), 3049-3067.

Questions?

Introduction to the BUGS language



BUGS

Spatial Capture-Recapture Workshop,

Cornell University, Ithaca, NY

April 2014

What is BUGS?

- Simple & flexible language to build (write down) sometimes very complex hierarchical models and analyze them in a Bayesian framework
- Implemented in a number of software packages
  - `WinBUGS` (Gilks et al. 1994)
  - `OpenBUGS` (Lunn et al. 2009)
  - `JAGS` (Plummer 2003)
- `WinBUGS` and `OpenBUGS` are standalone software
- `JAGS` has command line interface
- All three usually best run from **R** using interface packages

What is BUGS?

All three applications:

- Let you describe a statistical model using the BUGS language
- Translates this description into an MCMC algorithm
- Run the algorithm according to your specifications to produce samples from the joint posterior distribution of all unknown quantities in the model
- Provide functionalities for processing of results, convergence monitoring and so forth
- If you are still not convinced how useful these applications are: Write your own MCMC algorithm!!

What is BUGS?

BUGS
Bayesian inference **U**sing **G**ibbs **S**ampling

WinBUGS
Bayesian inference **U**sing **G**ibbs **S**ampling – **Win**dows based
[no longer being developed, not open source]

OpenBUGS
Bayesian inference **U**sing **G**ibbs **S**ampling – **Open** source
[currently active branch of the original BUGS project]

JAGS
Just **A**nother **G**ibbs **S**ampler
[separate project but uses essentially the same language]

What is BUGS?

The term '**BUGS**' will be used interchangeably

- The software `WinBUGS`/`OpenBUGS`
- The language used by `WinBUGS`/`OpenBUGS`/`JAGS`

From Kèry & Schaub book (2012):

"For most ecologists, WinBUGS is simply an ingenious MCMC blackbox. The analyst communicates with the MCMC engine by providing a data set and describing a statistical model for it using a simple and effective model definition language, the BUGS language (Gilks et al., 1994)."

Key components of BUGS language

- Nodes are the building blocks of BUGS
- Deterministic nodes:
  - Assigned with `<-`
  - Deterministic functions of other (parent) nodes
  - e.g. `x <- a + b`
- Stochastic nodes:
  - Assigned with `~`
  - Random variables, coming from a distribution
  - e.g. `x ~ dpois(lambda)`
- Data
  - Generally stochastic nodes
  - Can also be fixed (parent) nodes, appearing only on the right side of equations
  - e.g. `a` and `b` (above) could be data

Key components of BUGS language

- Nodes cannot be assigned in the form of vectors, matrices, arrays
- Almost all assignments have to happen element-wise in the form of **for-loops**

```
for (i in 1:M){
  mu[i]<-alpha + beta * VAR[i]
}
```

- Whereas in R we could simply write

```
mu<-alpha + beta * VAR
```

- An exception is the Multivariate distribution:

```
X[]~dmultinom(n, p[])
```

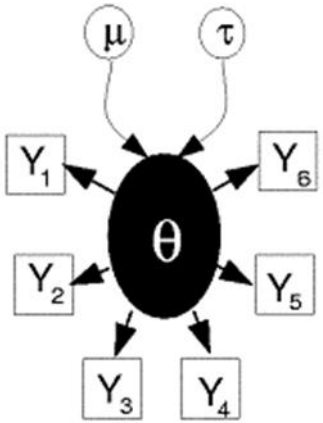
- BUGS is a declarative language, i.e. order of statements does (mostly) not matter
- Math functions and statistical distributions are listed in manuals

BUGS vs. glmr

- R functions `lmer()`/`glmer()` in package `lme4` most widely used to fit mixed models in ecology
- Mixed models = hierarchical models
- BUGS is much more flexible than glmer
- BUGS language a natural treatment of hierarchical models
- Bayesian implementation in BUGS allows focus on realizations of latent variables

Questions?

WinBUGS as a stand-alone application



BUGS

Spatial Capture-Recapture Workshop,

Cornell University, Ithaca, NY

April 2014

BUGS and R

- We will almost exclusively be using the R packages to call WinBUGS/JAGS from within R
- Important to get familiar with the workings of WinBUGS directly (*if anything, just to see how much more practical it is to use R*)

(1) Ingredients of a typical BUGS analysis

(2) Demo - Fit a simple model (linear regression)

Ingredients of a BUGS analysis

- Write the model
- Compile the data
- Provide (or generate) initial values
- Define parameters to be monitored

- Run model

- Look at output

Ingredients of a BUGS analysis

Write the model

e.g. consider a simple linear regression where observations, `y`, are a function of some covariate, `cov`

$$y_j \sim Normal(\mu_j, \sigma^2)$$
$$\mu_j = \alpha + \beta * cov_j$$

Ingredients of a BUGS analysis

$$y_j \sim Normal(\mu_j, \sigma^2)$$
$$\mu_j = \alpha + \beta * cov_j$$

Write the model

e.g. consider a simple linear regression where observations, `y`, are a function of some covariate, `cov`

In **R** we would simply write

```
lm(y ~ cov)
```

where α (intercept), β (slope) and σ^2 are *parameters* to be estimated *but* R takes care of that 'under the hood'.

Ingredients of a BUGS analysis

$$y_j \sim Normal(\mu_j, \sigma^2)$$
$$\mu_j = \alpha + \beta * cov_j$$

Write the model

e.g. consider a simple linear regression where observations, `y`, are a function of some covariate, `cov`

In **BUGS** we have to be more algebraically explicit

```
# Likelihood
for(j in 1:J){
  y[j] ~ dnorm(mu[j],tau)
  mu[j] <- alpha + beta * cov[j]
}
```

where α (intercept), β (slope) and σ^2 are *parameters* to be estimated *but* R takes care of that 'under the hood'.

J = number of observations (sites, individuals etc…)

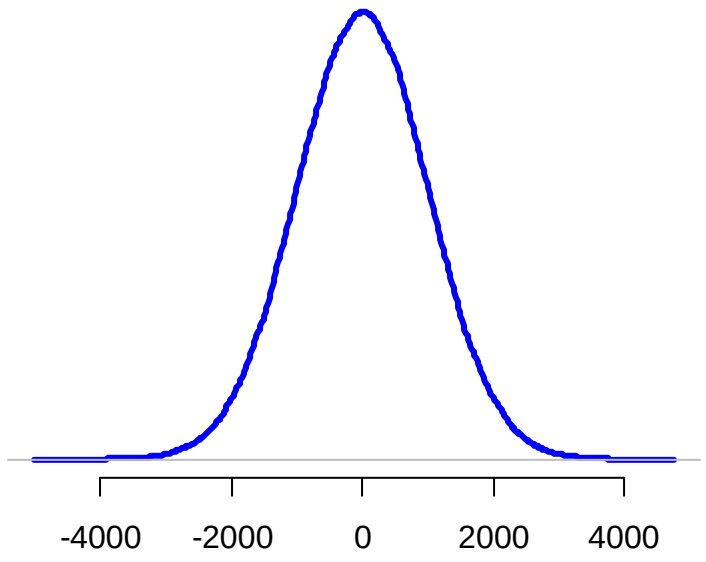
Ingredients of a BUGS analysis

$$y_j \sim Normal(\mu_j, \sigma^2)$$
$$\mu_j = \alpha + \beta * cov_j$$

Write the model

In the **BUGS** we also have to specify prior distributions for our parameters

```
# Priors
  alpha ~ dnorm(0,0.001)
  beta ~ dnorm(0,0.001)
  sigma ~ dunif(0,100)
  tau <- 1/(sigma*sigma)
```



BUGS uses precision (inverse of the variance)

e.g. normal with mean 0 and σ^2= 1/0.001 = 1000

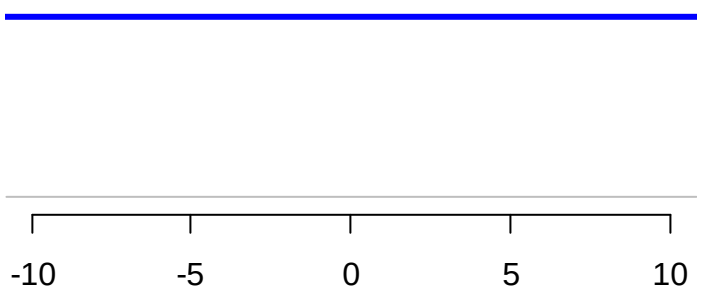
Ingredients of a BUGS analysis

$$y_j \sim Normal(\mu_j, \sigma^2)$$
$$\mu_j = \alpha + \beta * cov_j$$

Write the model

In the **BUGS** we also have to specify prior distributions for our parameters

```
# Priors
  alpha ~ dnorm(0,0.001)
  beta ~ dnorm(0,0.001)
  sigma ~ dunif(0,100)
  tau <- 1/(sigma*sigma)
```



BUGS uses precision (inverse of the variance)

e.g. normal with mean 0 and σ^2= 1/0.001 = 1000

Ingredients of a BUGS analysis

Write the model

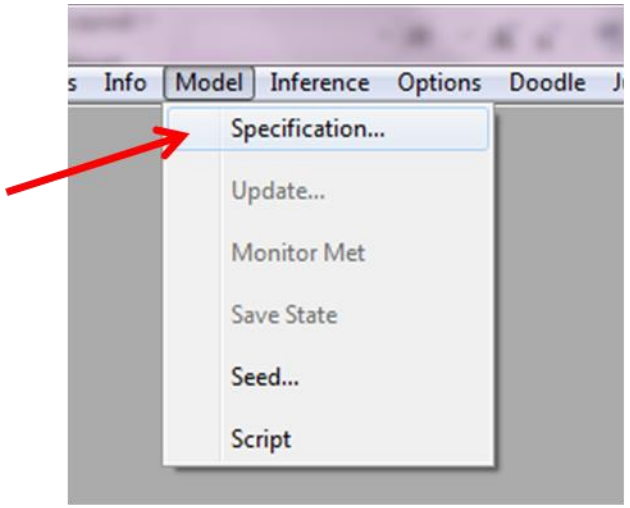
```
model{
# Priors
  alpha ~ dnorm(0,0.001)
  beta ~ dnorm(0,0.001)
  sigma ~ dunif(0,100)
  tau <- 1/(sigma*sigma)
# Likelihood
  for(j in 1:J){
    y[j] ~ dnorm(mu[j],tau)
   mu[j] <- alpha + beta * cov[j]
  } #end j
} #end model
```

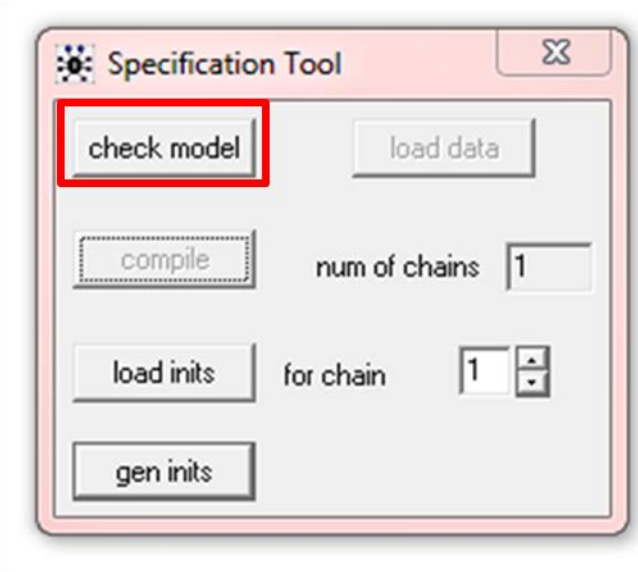
$$y_j \sim Normal(\mu_j, \sigma^2)$$
$$\mu_j = \alpha + \beta * cov_j$$

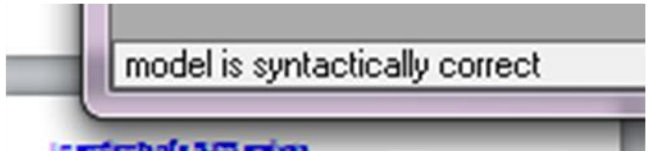
Ingredients of a BUGS analysis

Write the model

```
model{
# Priors
  alpha ~ dnorm(0,0.001)
  beta ~ dnorm(0,0.001)
  sigma ~ dunif(0,100)
  tau <- 1/(sigma*sigma)
# Likelihood
  for(j in 1:J){
    y[j] ~ dnorm(mu[j],tau)
   mu[j] <- alpha + beta * cov[j]
  } #end j
} #end model
```







Ingredients of a BUGS analysis

- Write the model
- **Compile the data**
  - Load the data into **WinBUGS**

ImDataWinBUGS

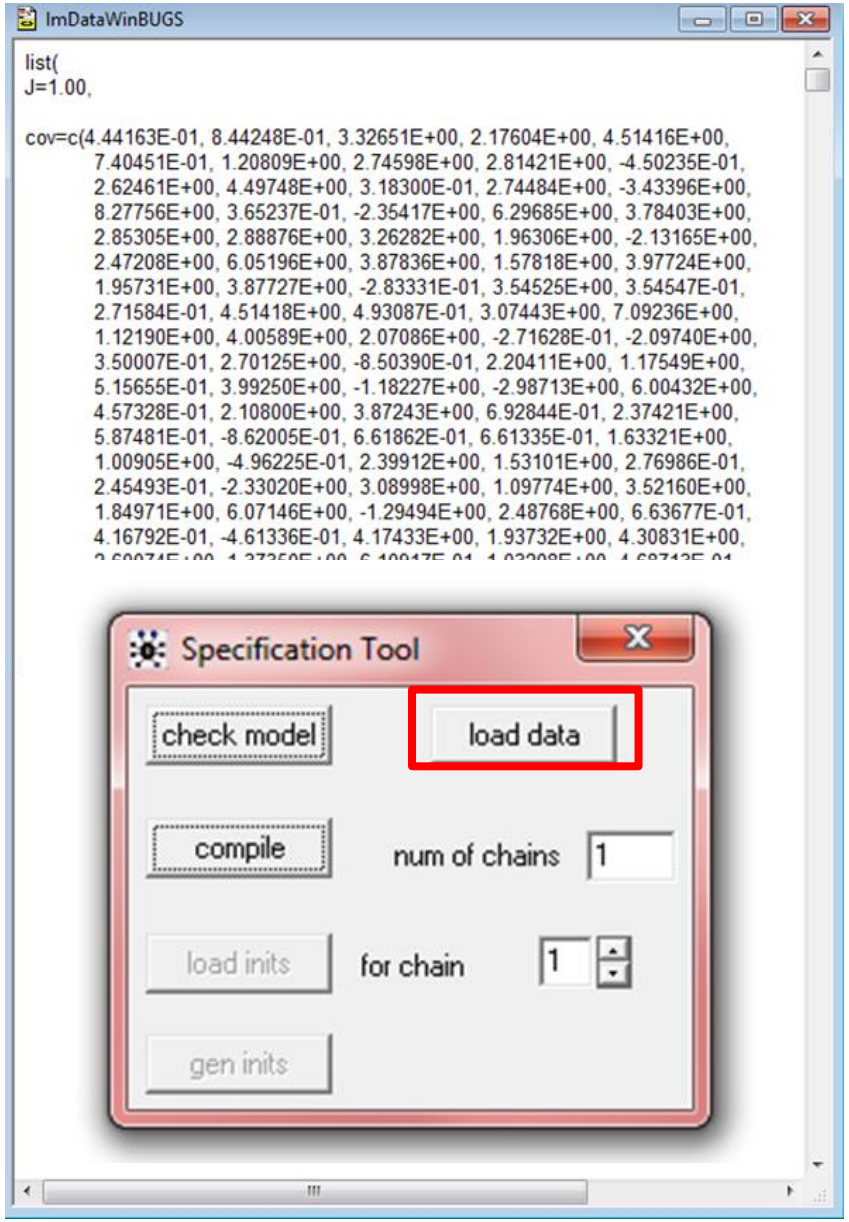
```
list(
J=1.00,

cov=c(4.44163E-01, 8.44248E-01, 3.32651E+00, 2.17604E+00, 4.51416E+00,
      7.40451E-01, 1.20809E+00, 2.74598E+00, 2.81421E+00, -4.50235E-01,
      2.62461E+00, 4.49748E+00, 3.18300E-01, 2.74484E+00, -3.43396E+00,
      8.27756E+00, 3.65237E-01, -2.35417E+00, 6.29685E+00, 3.78403E+00,
      2.85305E+00, 2.88876E+00, 3.26282E+00, 1.96306E+00, -2.13165E+00,
      2.47208E+00, 6.05196E+00, 3.87836E+00, 1.57818E+00, 3.97724E+00,
      1.95731E+00, 3.87727E+00, -2.83331E-01, 3.54525E+00, 3.54547E-01,
      2.71584E-01, 4.51418E+00, 4.93087E-01, 3.07443E+00, 7.09236E+00,
      1.12190E+00, 4.00589E+00, 2.07086E+00, -2.71628E-01, -2.09740E+00,
      3.50007E-01, 2.70125E+00, -8.50390E-01, 2.20411E+00, 1.17549E+00,
      5.15655E-01, 3.99250E+00, -1.18227E+00, -2.98713E+00, 6.00432E+00,
      4.57328E-01, 2.10800E+00, 3.87243E+00, 6.92844E-01, 2.37421E+00,
      5.87481E-01, -8.62005E-01, 6.61862E-01, 6.61335E-01, 1.63321E+00,
      1.00905E+00, -4.96225E-01, 2.39912E+00, 1.53101E+00, 2.76986E-01,
      2.45493E-01, -2.33020E+00, 3.08998E+00, 1.09774E+00, 3.52160E+00,
      1.84971E+00, 6.07146E+00, -1.29494E+00, 2.48768E+00, 6.63677E-01,
      4.16792E-01, -4.61336E-01, 4.17433E+00, 1.93732E+00, 4.30831E+00,
      2.60074E+00, 1.37350E+00, 6.10917E-01, 1.03208E+00, 4.68713E-01,
      1.69073E+00, 5.14632E+00, -7.36234E-01, 2.03010E+00, -1.60569E+00,
      3.03504E+00, -4.86507E-01, 1.75153E+00, 3.17853E+00, 3.66183E+00),

y=c(1.74557E+00, -1.08949E+00, -4.65828E+00, -2.92564E+00, -6.71363E+00,
     -1.78588E-01, -1.99015E+00, -1.76839E+00, -4.17746E+00, 5.68582E-01,
     -2.65280E+00, -8.88954E+00, -1.21450E+00, -1.02117E+00, 5.33981E+00,
     -1.11762E+01, 5.60563E-01, 4.09931E+00, -8.60344E+00, -7.46559E+00,
     -3.50997E+00, -4.87551E+00, -5.20630E+00, -3.10609E+00, 4.92278E+00,
     -3.80158E+00, -8.29335E+00, -3.96518E+00, -2.90226E+00, -7.03333E+00,
     -2.02741E+00, -4.98181E+00, 3.98591E-01, -4.94813E+00, -1.55788E+00,
     -1.36611E+00, -7.92118E+00, -1.51620E+00, -5.75739E+00, -9.33248E+00,
     -2.07801E+00, -5.88738E+00, -3.65116E+00, 1.51620E+00, 4.77925E+00,
     1.65869E+00, -3.60767E+00, 1.92081E+00, -4.16922E+00, -1.32918E+00,
     -8.36141E-01, -5.43389E+00, 2.19496E+00, 6.11884E+00, -9.76463E+00,
     2.75207E-01, -4.60923E+00, -4.69361E+00, 1.45930E+00, -3.44175E+00,
     -1.12075E+00, 2.26578E+00, 1.47476E+00, -7.94244E-01, -1.29542E+00,
     -7.00901E-01, 1.03747E+00, -4.34648E+00, -5.14464E-01, 1.21122E+00,
     -1.73986E+00, 2.00026E+00, -3.17739E+00, -1.05735E+00, -4.51124E+00,
     -2.97517E+00, -9.99763E+00, 3.17908E+00, -1.39179E+00, -1.63925E+00,
     1.80058E-01, 7.46877E-02, -5.56049E+00, -6.52885E-01, -5.18036E+00,
     -1.89908E+00, -3.30626E+00, 6.08268E-02, -1.28825E+00, 4.14088E-01,
     -2.49872E-01, -1.04302E+01, 4.74155E-02, -2.63101E+00, 5.00566E+00,
     -3.86103E+00, -1.22310E+00, -1.10954E+00, -4.39841E+00, -4.61790E+00)
```

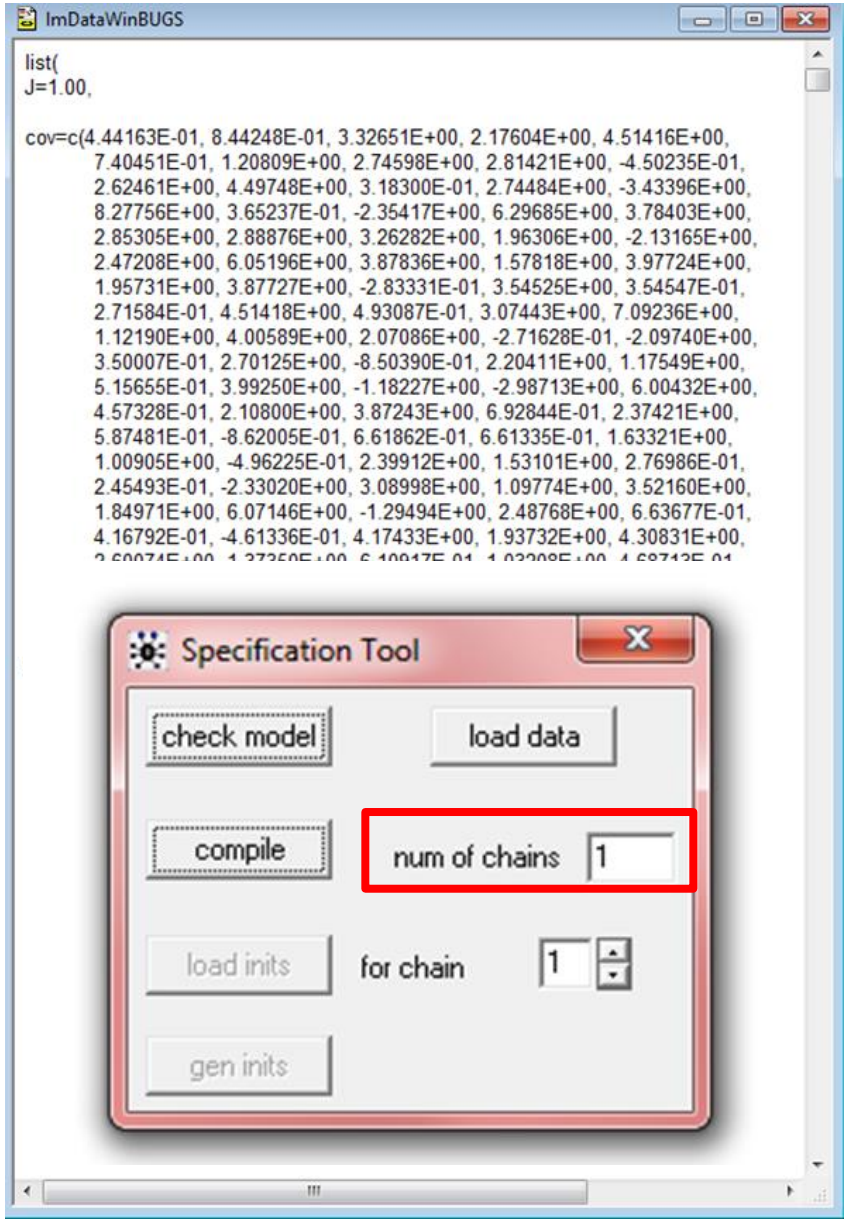
Ingredients of a BUGS analysis

- Write the model
- **Compile the data**
  - Load the data into **WinBUGS**



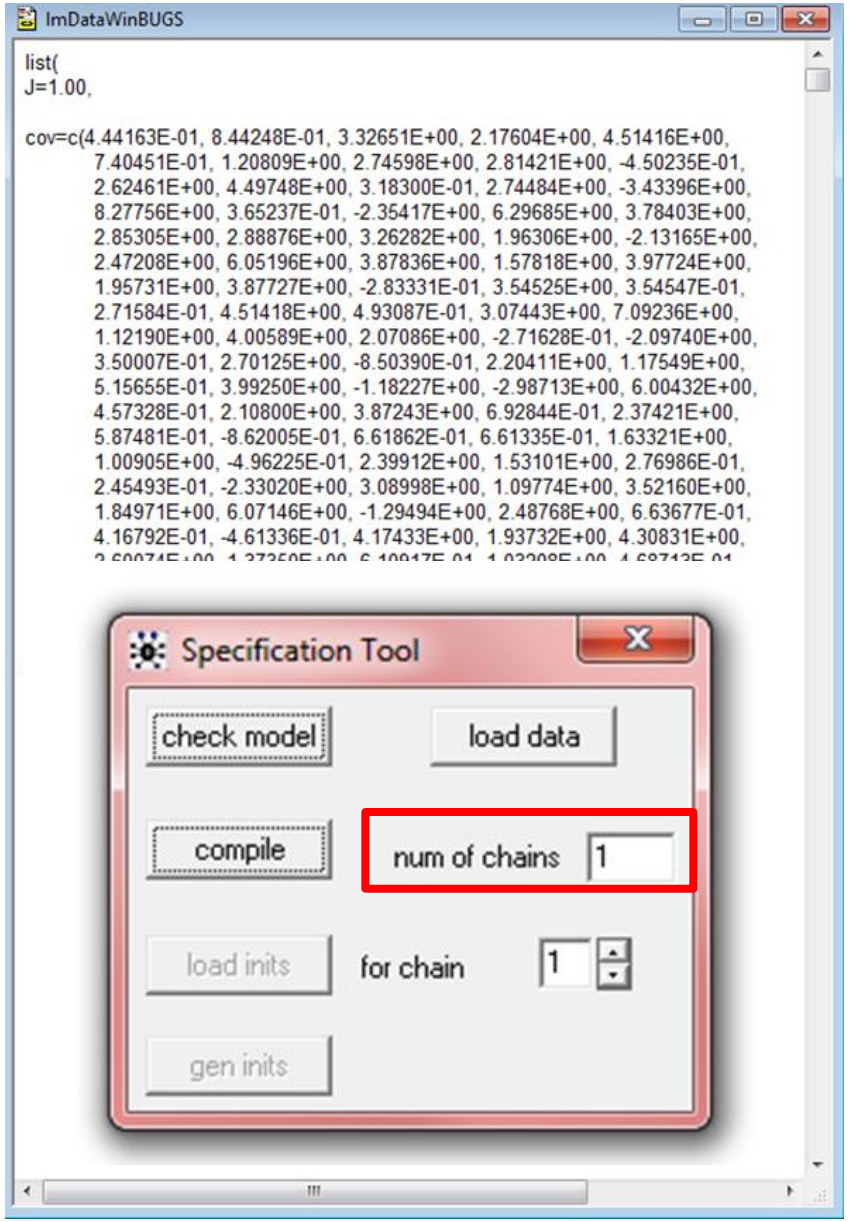
Ingredients of a BUGS analysis

- Write the model
- **Compile the data**
  - Load the data into **WinBUGS**
  - Select the number of parallel chains



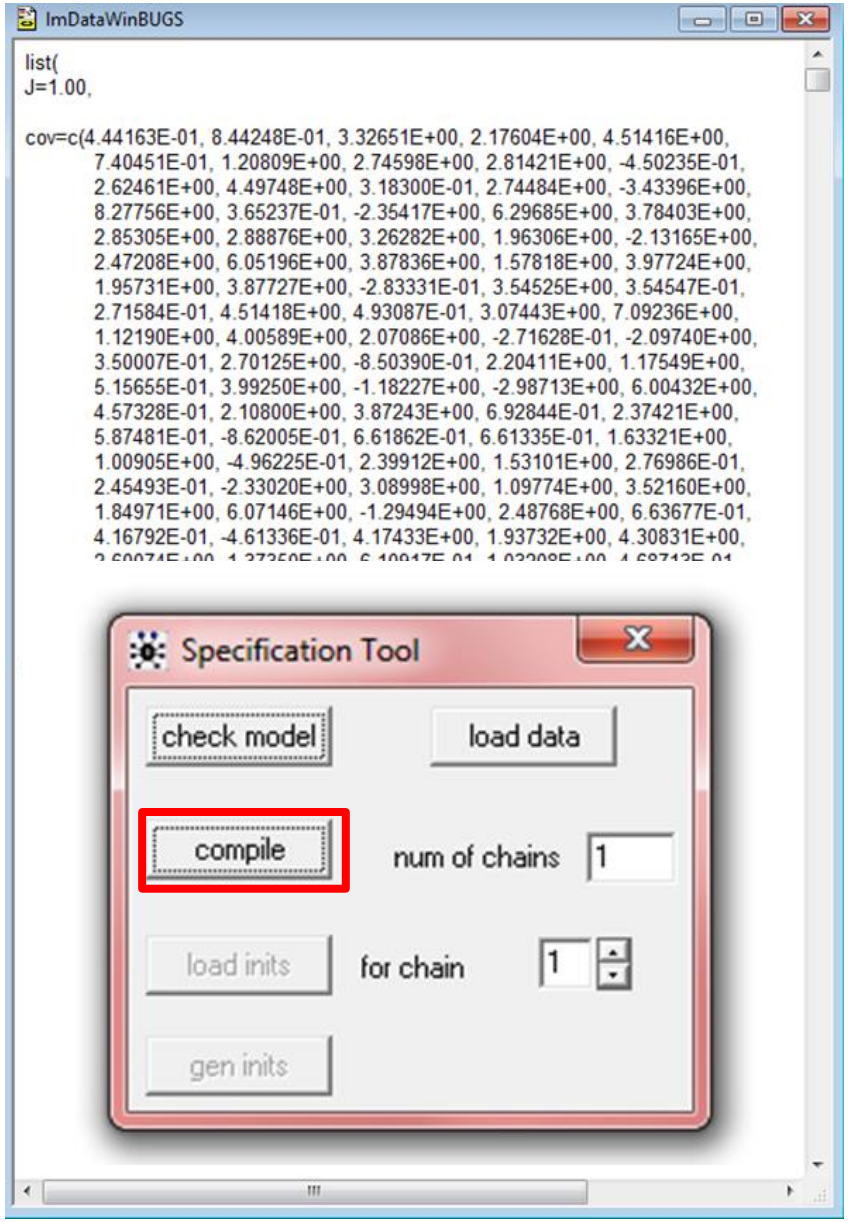
Ingredients of a BUGS analysis

- Write the model
- **Compile the data**
  - Load the data into **WinBUGS**
  - Select the number of parallel chains



Ingredients of a BUGS analysis

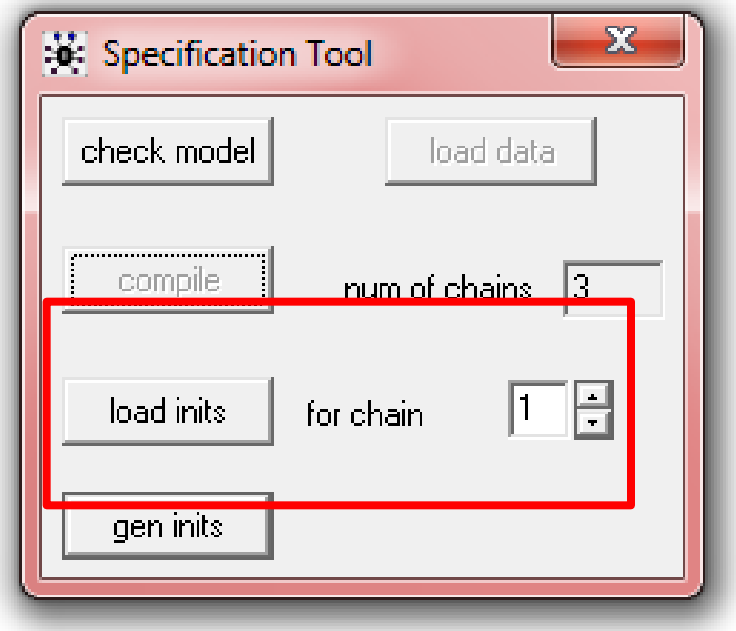
- Write the model
- **Compile the data**
  - Load the data into **WinBUGS**
  - Select the number of parallel chains
  - Compile the model and data



Ingredients of a BUGS analysis

- Write the model
- Compile the data
- **Provide (or generate) initial values**

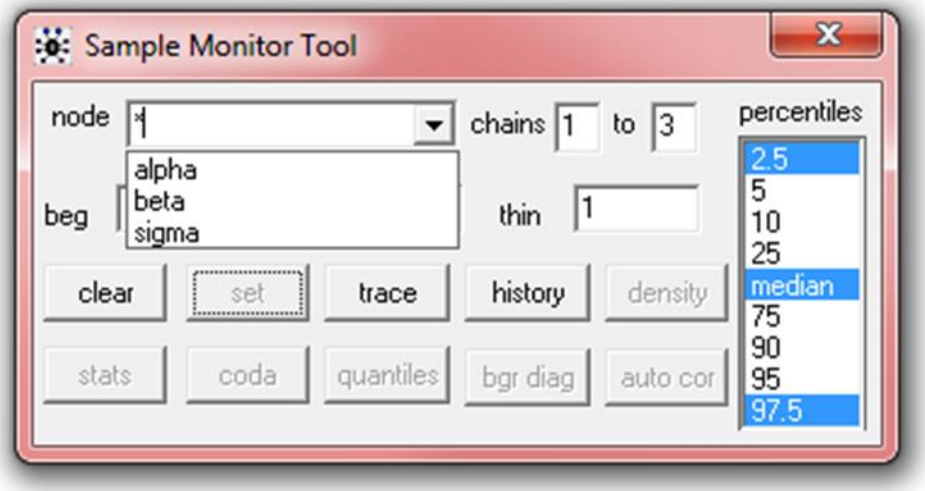
```
# initial values ('inits')
list(alpha = 0, beta = 1, sigma = 1)
list(alpha = 0, beta = 1, sigma = 10)
list(alpha = 0, beta = 1, sigma = 100)
```



Ingredients of a BUGS analysis

- Write the model
- Compile the data
- Provide (or generate) initial values
- **Determine parameters to be monitored**

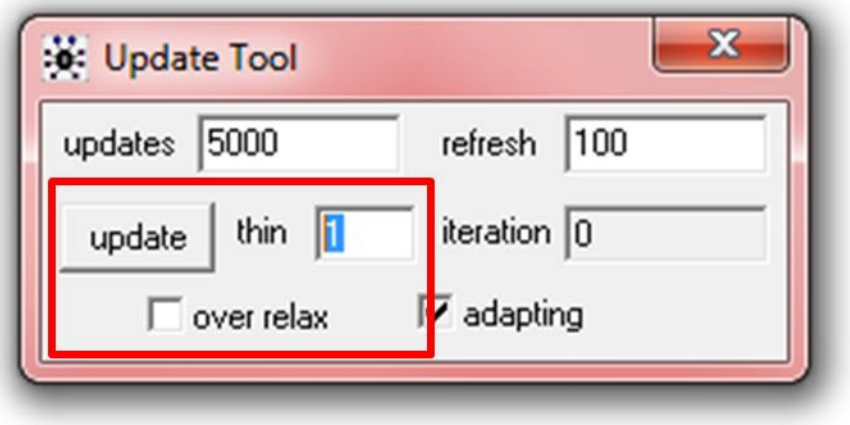
* *Monitors all named nodes*



Ingredients of a BUGS analysis

- Write the model
- Compile the data
- Provide (or generate) initial values
- Determine parameters to be monitored

- **Run model**
  - Select the number of iterations (updates)
  - Hit update
  - Viola!!!



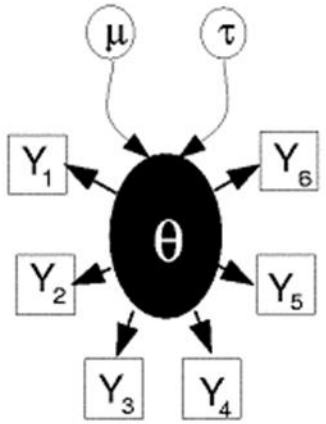
Ingredients of a BUGS analysis

- Write the model
- Compile the data
- Provide (or generate) initial values
- Determine parameters to be monitored

- Run model

- Look at output

Lets do it!



BUGS

$$y_j \sim Normal(\mu_j, \sigma^2)$$
$$\mu_j = \alpha + \beta * cov_j$$

Data simulated using: $\alpha = 0.5,\ \beta = -1.5, \sigma^2 = 1$

Combined: lmWinBUGS.odc

Model: lmModelWinBUGS.odc

Data: lmDataWinBUGS.odc

Inits: lmInitsWinBUGS.odc

Questions?

Accessing BUGS through R

Spatial Capture-Recapture Workshop,

Cornell University, Ithaca, NY

April 2014

Accessing BUGS through R

You can open `IntroBUGSscript1.R` and follow along if you like!

Generate data

Model 1:

$$y_j \sim Normal(\mu_j, \sigma^2)$$
$$\mu_j = \alpha + \beta * COV_j$$

```
###### generate data for a normal regression
J=100                    #number of sampling points
cov<-rnorm(J, 2,2)       #covariate
alpha<-0.5               #intercept
beta<--1.5               #coefficient
mu<-alpha+beta*cov       #expected value
sd<-1
y<-rnorm(J,mu,sd)        #observations

##analyze data using lm()
rmod<-lm(y~cov)
```

Ingredients of a BUGS analysis

- Write the model
- Compile the data
- Write an initial values function
- Determine parameters to be monitored

The same for all packages/programs

- Run model
- Look at output

Package/program-specific

Write the model

```
model{
# Priors
  alpha ~ dnorm(0,0.001)
  beta ~ dnorm(0,0.001)
  sigma ~ dunif(0,100)
  tau <- 1/(sigma*sigma)
# Likelihood
  for(j in 1:J){
    y[j] ~ dnorm(mu[j],tau)
   mu[j] <- alpha + beta * cov[j]
  } #end j
} #end model
```

$$y_j \sim Normal(\mu_j, \sigma^2)$$
$$\mu_j = \alpha + \beta * VAR_j$$

Write the model to a text file!

```
sink("binomGLMM.txt")
cat("
model{
# Priors
  alpha ~ dnorm(0,0.001)
  beta ~ dnorm(0,0.001)
  sigma ~ dunif(0,100)
  tau <- 1/(sigma*sigma)
# Likelihood
  for(j in 1:J){
    y[j] ~ dnorm(mu[j],tau)
   mu[j] <- alpha + beta * cov[j]
  } #end j
} #end model
",fill = TRUE)
sink()
```

$$y_j \sim Normal(\mu_j, \sigma^2)$$
$$\mu_j = \alpha + \beta * VAR_j$$

Data, initial values, parameters

Data

```
data<-list(J=J, cov=cov, y=y)
```

Initial values

- Supply initial values only for unobserved stochastic nodes
- Initial values have to fall within the range of support of the prior

```
inits<-function(){list(alpha=runif(1),
                       beta=runif(1),
                       sigma=runif(1,0,5))}
```

Parameters

```
params<-c("alpha", "beta", "sigma")
```

Run the model

- WinBUGS with `R2WinBUGS`
- OpenBUGS with `R2WinBUGS`/`BRugs`
- JAGS with `R2JAGS` or `rjags`
- Information needed:
  - Data, initial values, parameters
  - Number of parallel chains
  - Number of iterations
  - Length of adaptive phase/burn-in
  - Thinning interval

```
bugs(data, inits, parameters, model.file, n.chains=3,
     n.iter=5000,n.burn=1000,n.thin=1,debug=T)
```

Look at the output

- Gelman-Rubin ($\hat{R}$) statistic, effective sample size
  - Included in R2WinBUGS and R2jags output
  - Called with extra commands for rjags output

- Trace plots – using `xyplot()` in package coda
  - Do chains look grassy?
  - Do they oscillate around the same mean?

- Posterior density plots – using densityplot() in package coda
  - Were parameters estimable?
  - Are posterior distributions truncated, skewed, multimodal, …?

Lets do it in R!

$$y_j \sim Normal(\mu_j, \sigma^2)$$

$$\mu_j = \alpha + \beta * cov_j$$

Demo

Data simulated using: $\alpha = 0.5,\ \beta = -1.5, \sigma^2 = 1$

Script: `IntroBUGSscript1.R`

Exercise

Script: `IntroBUGSscript2.R`

Contains code to simulate data and write model

Your task: Complete steps to run Poisson regression in JAGS

(HINT: `log()` and `dpois()`)

Questions?

Mixed models in the BUGS language (random effects)

Spatial Capture-Recapture Workshop,

Cornell University, Ithaca, NY

April 2014

Random effects?

- Some parameters themselves may be realizations of a random process
- Lets consider a *random intercept* model
- i.e. intercepts come from a common distribution

	GLM (fixed effect)	**GLMM (mixed effect)**
Stochastic part	$y_j \sim Normal(\mu_j, \sigma^2)$	$y_j \sim Normal(\mu_j, \sigma^2)$
Linear predictor	$\mu_j = \alpha + \beta * cov_j$	$\mu_j = \alpha_k + \beta * cov_j$
Hyper parameters	-------	$\alpha_k \sim Normal(\mu_\alpha, \sigma_\alpha^2)$

Random effects?

- Some parameters themselves may be realizations of a random process
- Lets consider a *random intercept* model
- i.e. intercepts come from a common distribution

	GLM (fixed effect)	**GLMM (mixed effect)**
Stochastic part	$y_j \sim Normal(\mu_j, \sigma^2)$	$y_j \sim Normal(\mu_j, \sigma^2)$
Linear predictor	$\mu_j = \alpha + \beta * cov_j$	$\mu_j = \alpha_k + \beta_k * cov_j$
Hyper parameters	-------	$\alpha_k \sim Normal(\mu_\alpha, \sigma_\alpha^2)$ $\beta_k \sim Normal(\mu_\beta, \sigma_\beta^2)$

Generate some data?

GLM

$$y_j \sim Normal(\mu_j, \sigma^2)$$
$$\mu_j = \alpha + \beta * cov_j$$

```
###### generate data for a normal regression
J = 100                          # number of sampling points
cov <- rnorm(J, 2,2)             # covariate
alpha <- 0.5                     # intercept
beta <- -1.5                     # coefficient
mu <- alpha+beta*cov             # expected value
sd <- 1
y <- rnorm(J,mu,sd)              # observations
```

Generate some data?

GLMM

$$y_j \sim Normal(\mu_j, \sigma^2)$$
$$\mu_j = \alpha_k + \beta * cov_j$$
$$\alpha_k \sim Normal(\mu_\alpha, \sigma_\alpha^2)$$

```
###### generate data for a random intercept normal regression
J = 200                                          # observations
grp = gl(10,20)                                  # group membership k=10
cov <- rnorm(J, 2,2)                             # covariate
mu.alpha <- 0.5                                  # hyper mean for intercept
sd.alpha <- 1                                    # hyper sd for intercept
alpha <- rnorm(10,mu.alpha,sd.alpha)             # group level (RE) alphas
beta <- -1.5                                     # coefficient
mu <-alpha+beta*cov                              #expected value
sd <-1
y <- rnorm(J,mu,1)                               #observations
```

Generate some data?

GLMM

$$y_j \sim Normal(\mu_j, \sigma^2)$$
$$\mu_j = \alpha_k + \beta * cov_j$$
$$\alpha_k \sim Normal(\mu_\alpha, \sigma_\alpha^2)$$

```
###### generate data for a random intercept normal regression
J = 200                                          # observations
grp = gl(10,20)                                  # group membership
cov <- rnorm(J, 2,2)                             # covariate
mu.alpha <- 0.5                                  # hyper mean for intercept
sd.alpha <- 1                                    # hyper sd for intercept
alpha <- rnorm(10,mu.alpha,sd.alpha)             # group level (RE) alphas
beta <- -1.5                                     # coefficient
mu <-alpha+beta*cov                              #expected value
sd <-1
y <- rnorm(J,mu,1)                               #observations
```

Generate some data?

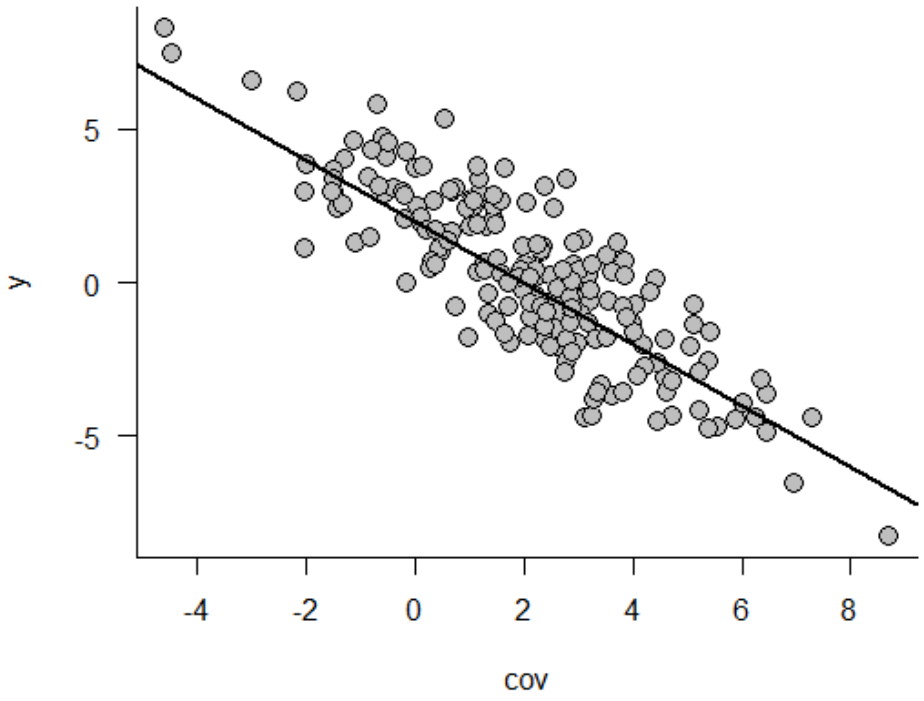
GLMM

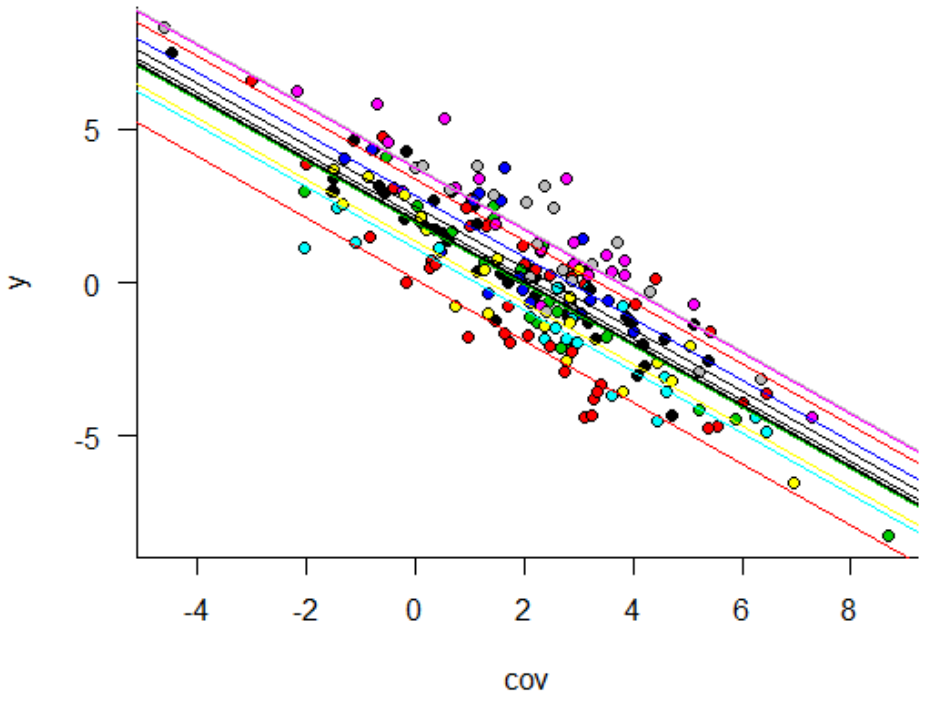
$$y_j \sim Normal(\mu_j, \sigma^2)$$
$$\mu_j = \alpha_k + \beta * cov_j$$
$$\alpha_k \sim Normal(\mu_\alpha, \sigma_\alpha^2)$$

```
###### generate data for a random intercept normal regression
J = 200                                          # observations
grp = gl(10,20)                                  # group membership
cov <- rnorm(J, 2,2)                             # covariate
mu.alpha <- 0.5                                  # hyper mean for intercept
sd.alpha <- 1                                    # hyper sd for intercept
alpha <- rnorm(10,mu.alpha,sd.alpha)             # group level (RE) alphas
beta <- -1.5                                     # coefficient
mu <-alpha[grp]+beta*cov                         # expected value
sd <-1
y <- rnorm(J,mu,1)                               # observations
```

Generate some data?

$$y_j \sim Normal(\mu_j, \sigma^2)$$
$$\mu_j = \alpha_k + \beta * cov_j$$
$$\alpha_k \sim Normal(\mu_\alpha, \sigma_\alpha^2)$$





Fitting a Normal GLM in the BUGS language

GLM

- Write the model - likelihood

```
# Likelihood for the GLM
  for(j in 1:J){
    y[j] ~ dnorm(mu[j],tau)
   mu[j] <- alpha + beta * cov[j]
  }
```

Fitting a Normal GLMM in the BUGS language

GLMM

- Write the model - likelihood

```
# Likelihood for the GLM
  for(j in 1:J){
    y[j] ~ dnorm(mu[j],tau)
   mu[j] <- alpha + beta * cov[j]
  }

# Likelihood for the GLMM
  for(j in 1:J){
    y[j] ~ dnorm(mu[j],tau)
   mu[j] <- alpha[grp[j]] + beta * cov[j]
  }
```

Fitting a Normal GLMM in the BUGS language

- Write the model - priors

```
# Priors GLM
  alpha ~ dnorm(0,0.001)
  beta ~ dnorm(0,0.001)
  sigma ~ dunif(0,100)
  tau <- 1/(sigma*sigma)
```

```
# Priors GLMM
  for(k in 1:n.gr){
    alpha[k] ~ dnorm(mu.alpha,tau.alpha)
  }
  mu.alpha ~ dnorm(0,0.001)
  tau.alpha <- 1/sd.alpha*sd.alpha
  sd.alpha ~ dunif(0,20)

  beta ~ dnorm(0,0.001)
  tau <- 1/(sigma*sigma)
  sigma ~ dunif(0,100)
```

Fitting a Normal GLMM in the BUGS language

- Write the model:

$$y_j \sim Normal(\mu_j, \sigma^2)$$
$$\mu_j = \alpha_k + \beta * cov_j$$
$$\alpha_k \sim Normal(\mu_\alpha, \sigma_\alpha^2)$$

```
model{
# Priors GLMM
  for(k in 1:n.gr){
    alpha[k] ~ dnorm(mu.alpha,tau.alpha)
  }
  mu.alpha ~ dnorm(0,0.001)
  tau.alpha <- 1/sd.alpha*sd.alpha
  sd.alpha ~ dunif(0,20)
  beta ~ dnorm(0,0.001)
  tau <- 1/(sigma*sigma)
  sigma ~ dunif(0,100)
# Likelihood
  for(j in 1:J){
    y[j] ~ dnorm(mu[j],tau)
   mu[j] <- alpha[grp[j]] + beta * cov[j]
  }
}
```

Fitting a Normal GLMM in the BUGS language

- Write the model:

$$y_j \sim Poisson(\mu_j)$$
$$log(\mu_j) = \alpha_k + \beta * cov_j$$
$$\alpha_k \sim Normal(\mu_\alpha, \sigma_\alpha^2)$$

```
model{
# Priors GLMM
  for(k in 1:n.gr){
    alpha[k] ~ dnorm(mu.alpha,tau.alpha)
  }
  mu.alpha ~ dnorm(0,0.001)
  tau.alpha <- 1/sd.alpha*sd.alpha
  sd.alpha ~ dunif(0,20)
  beta ~ dnorm(0,0.001)
  #tau <- 1/(sigma*sigma)
  #sigma ~ dunif(0,100)
# Likelihood
  for(j in 1:J){
    y[j] ~ dpois(mu[j])
   log(mu[j]) <- alpha[grp[j]] + beta * cov[j]
  }
}
```

Fitting a Normal GLMM in the BUGS language

- Write the model:

$$y_j \sim Bernoulli(\mu_j)$$
$$logit(\mu_j) = \alpha_k + \beta * cov_j$$
$$\alpha_k \sim Normal(\mu_\alpha, \sigma_\alpha^2)$$

```
model{
# Priors GLMM
  for(k in 1:n.gr){
    alpha[k] ~ dnorm(mu.alpha,tau.alpha)
  }
  mu.alpha ~ dnorm(0,0.001)
  tau.alpha <- 1/sd.alpha*sd.alpha
  sd.alpha ~ dunif(0,20)
  beta ~ dnorm(0,0.001)
  #tau <- 1/(sigma*sigma)
  #sigma ~ dunif(0,100)
# Likelihood
  for(j in 1:J){
    y[j] ~ dbern(mu[j])
   logit(mu[j]) <- alpha[grp[j]] + beta * cov[j]
  }
}
```

Lets do it in R!

Demo

Normal GLM -> Normal GLMM

Script: `normGLMM.R`

$$y_j \sim Normal(\mu_j, \sigma^2)$$

$$\mu_j = \alpha + \beta * cov_j$$

Exercise

Script: `binomGLMMEX.R`

Contains code to simulate data and write model

Your task: Complete steps to run Binomial GLMM in JAGS

$$y_j \sim Bernoulli(\mu_j)$$

$$\mu_j = \alpha + \beta * cov_j$$

(HINT: `logit()` and `dbern()`)

Questions?

Non-standard GLMMs – an occupancy model!

- Occupancy state (occupied: $z_i = 1$ else $z_i = 0$)
  - rarely observed perfectly

$$y_i \neq z_i$$

  - a latent variable and a discrete random effects drawn from a Bernoulli distribution

$$z_i \sim Bernoulli(\psi)$$

  - data y_i are imperfect observations of the true occupancy state from K visits

$$y_i \sim Binomial(K, z_i * p_i)$$

- Aim to estimate N, the number of occupied sites: $N = \sum_{i=1}^{M} z_i$

Non-standard GLMMs – an occupancy model!

- A standard occupancy model:

$$z_i \sim Bernoulli(\psi)$$
$$y_i \sim Binomial(K, p * z_i)$$

- We can also account for site level heterogeneity in detection, e.g. detection decreases with increasing vegetation cover (`cover`)

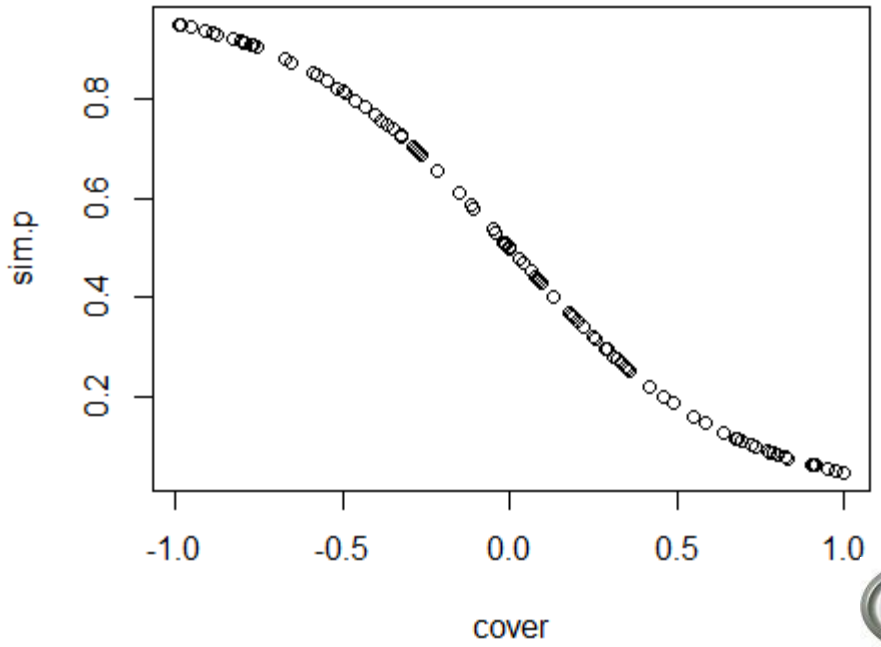
$$z_i \sim Bernoulli(\psi)$$
$$y_i \sim Binomial(K, \boldsymbol{p_i} * z_i)$$
$$logit(p_i) = \alpha + \beta * \text{cover}_i$$

Non-standard GLMMs – an occupancy model!

```
# Generate some data:
 set.seed(1235)
 M <- 100
 sim.psi <- 0.6
 K <- 3
 cover <- runif(M,-1,1)
 alpha <- 0
 beta <- -3
 sim.p <- plogis(alpha + beta*cover)
 plot(cover,sim.p)
 z.vec <- rbinom(M,1,sim.psi)
 p.eff <- sim.p * z.vec
 y <- rbinom(M,K,p.eff)
 N.true <- sum(z.vec)
 N.obs <- sum(y>0)

 (N.true) # 53 or 0.53
 (N.obs)  # 42 or 0.42
```

$$z_i \sim Bernoulli(\psi)$$
$$y_i \sim Binomial(K, \boldsymbol{p_i} * z_i)$$
$$logit(p_i) = \alpha + \beta * \text{cover}_i$$



Non-standard GLMMs – an occupancy model!

```
sink("occMod.txt")
cat("
model{
# Priors
  psi ~ dunif(0,1)
  alpha.p ~ dnorm(0,0.001)
  beta.p ~ dnorm(0,0.001)
# Likelihood
  for(m in 1:M){
    z[m] ~ dbern(psi)
    y[m] ~ dbin(mu.p[m] * z[m],K)
    logit(mu.p[m]) <- alpha.p + beta.p * cover[m]
  } #end j
N <- sum(z[]) # sum the z's to derive N!!!
} #end model
",fill = TRUE)
sink()
```

$$z_i \sim Bernoulli(\psi)$$
$$y_i \sim Binomial(K, p_i * z_i)$$
$$logit(p_i) = \alpha + \beta * \text{cover}_i$$

$$N = \sum_{i=1}^{M} z_i$$

Lets do it in R!

$$z_i \sim Bernoulli(\psi)$$

$$y_i \sim Binomial(K, p_i * z_i)$$

$$p_i = \alpha + \beta * \text{cover}_i$$

Exercise

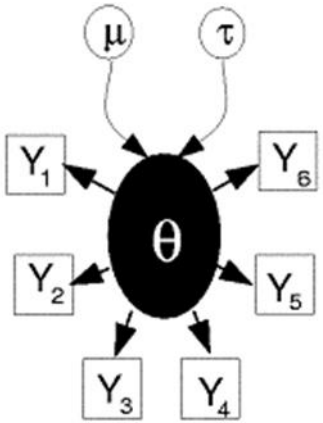
Script: `occupancyEX.R`

Contains code to simulate data and write model

Your task: Complete steps to fit the occupancy model in JAGS

Questions?

Summary & Outlook



BUGS

Spatial Capture-Recapture Workshop,

Cornell University, Ithaca, NY

April 2014

Summary

- Many ways to implement MCMC
  - Win/Open BUGS, JAGS
  - R2WinBUGS, R2jags, rjags

- Beyond BUGS
  - BUGS language -> flexibility, transparency
  - Natural means to deal with latent variables/random effects
  - Do Bayesian analysis without writing samplers
  - Often easier or even only way to code more complex hierarchical models

Beyond ‘simple’ BUGS

- Special features in Win/OpenBUGS
  - WinBUGS Jump interface: Reversible jump MCMC
  - Autologistic models

- Alternative MCMC software
  - Stan (mc-stan.org): faster, different algorithms, but still somewhat limited applicability
  - SAS/STAT/R MCMC Procedure and other applications
  - Do-it-yourself in R

Questions?

Categorical variables in the BUGS language

Model 2:

$$y_j \sim Normal(\mu_j, \sigma^2)$$

$$\mu_j = \alpha_{COV_j}$$

for **COV** = {1, 2, 3, 1, 1, 2, 1…}

Means parameterization

OR

$$\mu_j = \alpha + \beta_2 * COV2_j + \beta_3 * COV3_j$$

For **COV2** = {0, 1, 0, 0, 0, 1, 0,…}

COV3 = {0, 0, 1, 0, 0, 0, 0,…}

Contrast parameterization

Categorical variables in the BUGS language

Alternative parameterizations for a categorical variable

```
# MEANS

# Priors
  for(g in 1:ngr){
    alpha[g] ~ dnorm(0,0.001)
  }

 sigma ~ dunif(0,100)
 tau <- 1/(sigma*sigma)

# Likelihood
  for(j in 1:J){
    y[j] ~ dnorm(mu[j],tau)
   mu[j] <- alpha[cov[j]]
  }
```

```
# CONTRAST/REFERENCE

# Priors
  for(g in 2:ngr){
    beta[g] ~ dnorm(0,0.001)
  }
  beta[1] <- 0
  alpha ~ dnorm(0,0.001)
  sigma ~ dunif(0,100)
  tau <- 1/(sigma*sigma)

# Likelihood
  for(j in 1:J){
    y[j] ~ dnorm(mu[j],tau)
   mu[j] <- alpha + beta[cov[j]]
  }
```

Non-standard GLMMs – detection covariate

Data

```
data<-list(M=M, cover=cover, y=y)
```

Initial values

Supply initial values only for unobserved stochastic nodes

Initial values have to fall within the range of support of the prior

```
inits<-function(){list(psi=runif(1,0,1), alpha.p=runif(1),
                       beta.p=runif(1), sigma=runif(1,0,5))}
```

Parameters

```
params<-c("psi","alpha", "beta", "sigma","N")
```