WinBUGS for Beginners

Gabriela Espino-Hernandez

Department of Statistics

UBC

July 2010

“A knowledge of Bayesian statistics is assumed…”

The content of this presentation is mainly based on WinBUGS manual

Introduction

- **BUGS**[1]: "**B**ayesian inference **U**sing **G**ibbs **S**ampling"
  - Project for Bayesian analysis using MCMC methods
  - It is not being further developed

- **WinBUGS**[1,2]
  - Stable version
  - Run directly from R and other programs

- **OpenBUGS**[3]
  - Currently experimental
  - Run directly from R and other programs
  - Running under Linux as LinBUGS

[1] *MRC Biostatistics Unit Cambridge,* [2] *Imperial College School of Medicine at St Mary's, London*
[3] *University of Helsinki, Finland*

WinBUGS

- Freely distributed
  http://www.mrc-bsu.cam.ac.uk/bugs/welcome.shtml
- Key for unrestricted use
  http://www.mrc-bsu.cam.ac.uk/bugs/winbugs/WinBUGS14_immortality_key.txt
- WinBUGS installation also contains:
  - Extensive user manual
  - Examples
- Control analysis using:
  - Standard windows interface
  - DoodleBUGS: Graphical representation of model
- A closed form for the posterior distribution is not needed
- Conditional independence is assumed
- Improper priors are not allowed

Inputs

- Model code
  - Specify data distributions
  - Specify parameter distributions (priors)
- Data
  - List / rectangular format
- Initial values for parameters
  - Load / generate

Model specification

```
model {
        statements to describe model in BUGS language
      }
```

- Multiple statements in a single line or one statement over several lines
- Comment line is followed by #
- Types of nodes
  1. *Stochastic*
     - Variables that are given a distribution
  2. *Deterministic / Logical*
     - Logical functions of other nodes
  3. *Constant*

Model specification

- Stochastic and deterministic nodes appears only once on the LHS of a statement (except in data transformation)
- Name nodes
  - Case sensitive
  - Characters allowed: letters, numbers and period
  - Start with a letter
  - No end with a period
  - Maximum 32 characters
- Numbers
  - Standard or exponential notation
  - Include decimal point in the exponential format
  - Notation allowed:

.0001	0.0001
1.0E-4	1.0e-4
1.E-4	1.0E-04

  - Not allowed: 1E-4

Model specification

- Stochastic nodes:
  - Assign distributions using ~ (*"is distributed as"*)
  - Help → WinBUGS user manual → Distributions
  - Handle 19 discrete/continuous univariate distributions:

    eg: Bernoulli, Binomial, Categorical, Negative Binomial, Poisson, Beta, Chi-squared, Exponential, Gamma, Normal, Pareto, Student-t, Uniform, Weibull, etc
  - Handle 5 discrete/continuous multivariate distributions:

    Multinomial, MVN, Multivariate Student-t, Dirichlet, Wishart
  - No missing data, except for multivariate normal data

Model specification

- Stochastic nodes:
  - Examples

```
for (j in 1:J) {
        y[j] ~ dbin(p[j], n[j])
}
```

```
for (i in 1:N) {
        z[i] ~ dnorm(mu, tau)
}
```

Note: tau = precision of normal distribution

```
for (i in 1:N) {
        Y[i, 1:M] ~ dmnorm(mu[], Omega[ , ])
}
```

Model specification

- Logical nodes:
  - Assign logical expressions using <- (*“to be replaced by”*)
  - Logical expressions can be built using:
    - Operators: +, -, *, /, unitary minus (-A)
    - Logical functions:

      Help → WinBUGS user manual → Model Specification → Logical nodes → Table I: Functions
  - Examples:

```
N <- K+2

mu[i] <- beta0 + beta * z[i] + b[i]

prec[1:2, 1:2] <- inverse( var.matrix[,] )

log( e[i, j] ) <-  inprod( beta[1:3] , X[i, j, 1:3 ] )
```

Model specification

- Logical nodes:
  - Cannot be given data or initial values (except when using the data transformation)
  - Link functions that can be used on the left hand side of logical nodes

    log logit cloglog probit

```
logit(mu[i]) <- beta0 + beta * z[i] + b[i]
```

Model specification

- Data transformation:
    - Data *y* available
    - Want to model *sqrt(y)*

```
for (i in 1:N) {
                z[i] <- sqrt(y[i])
                z[i] ~ dnorm(mu, tau)
}
```

Model specification

- Arrays and indexing
  - Vectors and matrices are indexed within square brackets *[ ]*
    - *n:m* represents *n,n+1,...,m*
    - *y[ ]* indicates all values of a vector *y*
    - *y[,3]* indicates all values of the *third* column of a two-dimensional array *y*

  - Range of the variable must be explicitly defined in multivariate nodes
    - *x**[1:K]** ~ dmnorm(mu**[ ]**, tau**[,]**)*

  - Up to three dimensions are currently allowed (?)

  - Multidimensional arrays are handled as one-dimensional arrays with a constructed index
    - *y**[** i, j **]***
    - *x**[** i, j, 1:3 **]***

Model specification

- Censoring
  - Interval censored: *y ~ ddist(theta)I(lower, upper)*
  - Right censored: *y ~ ddist(theta)I(lower,)*
  - Left censored: *y ~ ddist(theta)I(,upper)*
  - The structure is only use if *y* has not been observed, otherwise the constraints will be ignored
  - If *y*, *theta*, *lower* and *upper* are not observed, then *lower* and *upper* must not be functions of *theta*

Data

- Available data and constants nodes are specified in a data file
- Format:
  - List (S-Plus/R format)
  - Rectangular: Data in arrays
- Whole array must be specified in a single file
- Missing values are represented as *NA*
- All variables in a data file must be defined in a model
- It is possible to load a mixture of list and rectangular format data for the same model (see Rats example)

Data: list format

```
list(
            data in a list format
     )
```

- Scalars and arrays are named and given values in a single structure
- No space between *list* and *(*
- Example:

```
list(
            xbar = 22, N = 3, T = 5,
            x = c(8.0, 15.0, 22.0, 29.0, 36.0),
            Y = structure(
                              .Data =  c(151, 199, 246, 283, 320,
                                             145, 199, 249, 293, 354,
                                             153, 200, 244, 286, 324),
                              .Dim = c(3,5) )
     )
```

Data: list format

- Example of 3 * 2 * 4 dimensional array:

```
1       2       3       4
5       6       7       8

9       10      11      12
13      14      15      16

17      18      19      20
21      22      23      24
```

```
list(
        Y = structure(
                        .Data =  c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,13, 14,
                                   15, 16, 17, 18, 19, 20, 21, 22, 23, 24),
                        .Dim = c(3, 2, 4) )
      )
```

Data: rectangular format

```
var1[ ]   var2[ ]  .....    varK[ ]
.....       .....       .....      .....
END
```

- Columns headed by array name
- Arrays of equal size
- Array names must have explicit brackets
- The first index position of any array must be empty
- End file with an '*END*' statement, followed by at least one blank line
- Example:

```
age[ ]  sex[ ]
26      0
52      1
....
END
```

Data: rectangular format

```
var1[ ]   var2[ ]  .....    varK[ ]
.....     .....    .....     .....
END
```

- Example of multi-dimensional arrays:

```
Y[,1]   Y[,2]   Y[,3]   Y[,4]   Y[,5]
151     199     246     283     320
145     199     249     293     354
147     214     263     312     328
.......
153     200     244     286     324
END
```

Initial values

- Required for all stochastic nodes (without available data)
- Initial values can be:
  - Load
    - Specify initial values following same format as Data
  - Generated
    - Use Specification tool

Air example

- Evaluate association between:
  - Respiratory illness and exposure to nitrogen dioxide (NO_2)
- Measurement error in the exposure
- Studied subjects: 103 children
- Data:

	Bedroom NO_2 level in ppb (Z)			
Respiratory illness (y)	**<20**	**20-40**	**40+**	**Total**
Yes	21	20	15	**56**
No	27	14	6	**47**
Total	**48**	**34**	**21**	**103**

y response variable.

Z observed exposure. 3 categories z_j ($j = 1, 2, 3$)

Air example

- Berkson measurement error model: Association between true exposure (***X***) and observed exposure (***Z***)

$$X_j = \alpha + \beta Z_j + \varepsilon_j$$

where α = 4.48,

β = 0.76 and

ε_j follows a normal distribution with zero mean and variance 81.14

- Observed exposure (Z_j) takes values 10, 30 or 50

Air example

- A logistic regression model is assumed

$$y_j \sim \text{Binomial}(p_j, n_j)$$

$$\text{logit}(p_j) = \theta_1 + \theta_2 X_j$$

where p_j = probability of respiratory illness for children in the j-th exposure group

θ_1 and θ_2 are the regression coefficients

- The regression coefficients are given vague independent normal priors

Air example

- Model

```
model {
        for(j in 1 : J) {

                # DISEASE MODEL
                y[ j ] ~ dbin(p[ j ], n[ j ])
                logit(p[j]) <- theta[1] + theta[2] * X[j]

                # MEASUREMENT ERROR MODEL
                X[ j ] ~ dnorm(mu[ j ], tau)
                mu[ j ] <- alpha + beta * Z[ j ]
                }

        # PRIORS
        theta[1] ~ dnorm(0.0, 0.001)
        theta[2] ~ dnorm(0.0, 0.001)
}
```

Air example

- Data

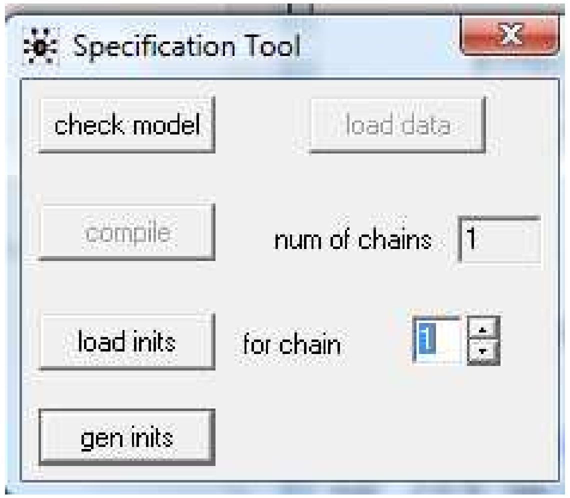
```
list(  J = 3, alpha = 4.48, beta = 0.76, tau = 0.01234,
       n = c(48, 34, 21),
       Z = c(10, 30, 50),
       y = c(21, 20, 15))
```

- Initial values

```
list(  theta = c(0.0, 0.0),  X = c(0.0, 0.0, 0.0))
```

Run model

Menu bar → Model → Specification tool



1. Check model

   "model is syntactically correct"

2. Load data

   "data loaded"

3. Specify num. of chains to be simulated

4. Compile model

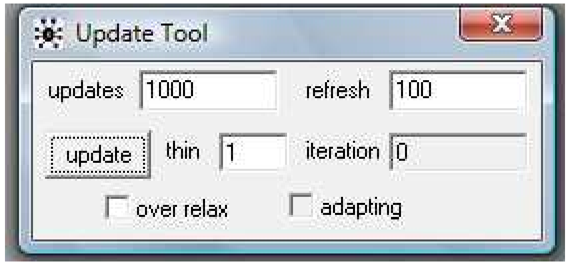
   "model compiled"

5. Initial values for each chain
   - Load *"model is initialized"*
   - Generate *"initial values generated, model initialized"*

Update model

Menu bar → Model → Update tool

1. Specify number of MCMC updates to be carried out


Update Tool

updates 1000 refresh 100

update thin 1 iteration 0

over relax adapting

- Data have not been stored
- Used for “burn-in” period

Monitor runs

Menu bar → Inference → Sample monitor tool

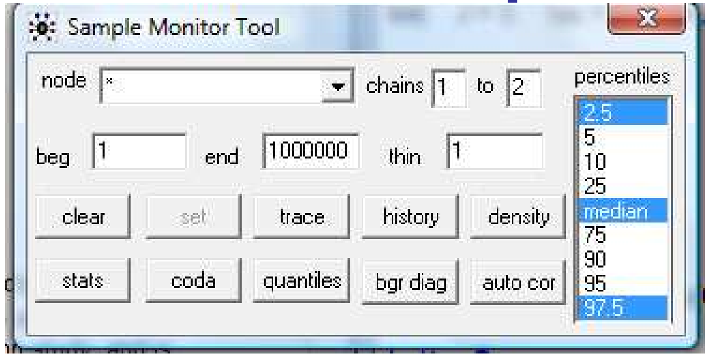
Monitor a single node (unknown quantity)

- Type name of node to be monitored
- Click on “set”
- Update model

Generate simulated values of all unknown quantities

- Repeat steps 1 and 2 for all nodes to be monitored
- Type * to select all the nodes
- Update model

Sample monitor tool

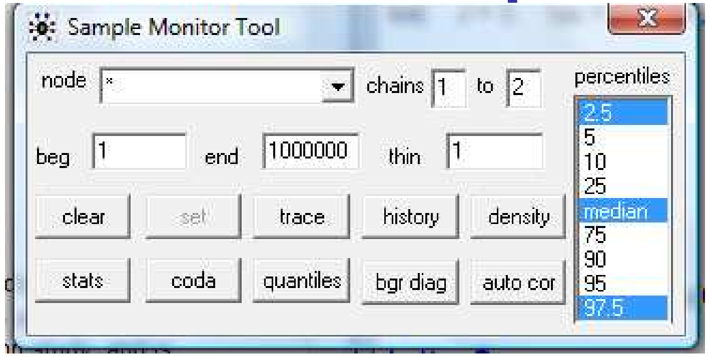


node: *
chains 1 to 2
beg 1 end 1000000 thin 1
percentiles: 2.5, 5, 10, 25, median, 75, 90, 95, 97.5

trace	Plots variable value against iteration number (dynamic)
history	Plots a complete trace
density	Plots a smoothed kernel density estimate / histogram
stats	Summary statistics
coda	ASCII representation of the monitored values for use in R
quantiles	Plots running mean and 95% CI against iteration num.
auto cor*	Plots/calculates the autocorrelation function of the variable out to lag-50

** Values can be listed by double-clicking on the plot followed by ctrl-left-mouse-click*

Sample monitor tool



- **bgr diag*** Plots/calculates the Gelman-Rubin convergence statistic.
  - Blue: Width of the 80% interval of the pooled runs
  - Green: Average width of the 80% intervals within the individual runs
  - Red: R = pooled / within
  - Interval widths are normalised to have an overall maximum of 1
  - R >1 if starting values are over-dispersed
  - Look for convergence of R to 1, and convergence of pooled and within interval

** Values can be listed by double-clicking on the plot followed by ctrl-left-mouse-click*

STATS

node	Name of the unknown quantity
mean	Posterior mean of the unknown quantity
sd	Posterior standard deviation of the unknown quantity
MC error	Computational accuracy of the mean
2.5%	Lower endpoint of the 95% credible interval
median	Posterior median
97.5%	Upper endpoint of the 95% credible interval
start	The starting simulation (after discarding the start-up)
sample	Number of simulations used to approximate the posterior distribution

Output options

- By default WinBUGS opens a new window for each output (stats, history, density, …)
- All output could be written into single log file:
  - Options menu → Output options → select “log”
- Access to the output:
  - Info menu → Open Log

CODA

- **CODA**: **C**onvergence **D**iagnostic and **O**utput **A**nalysis.

- Files produced:
  - Output file for each chain: ***CODA for chain N*** or (***.out*** file)
    - Contains iteration number and value
  - Description file: ***CODA index*** or (***.ind*** file)
    - Indicates which lines of the ***CODA for chain N*** file correspond to which variable

- Save files:
  - Menu bar → File → Save as
  - Select *Plain Text (.txt)* in *Save as type*
  - Give a file name, enclose it in quotes, and save it
  - Example:
    - Output files: **"AirChain1.out"** and **"AirChain2.out"**
    - Description file: **"Air.ind"**

Analysis using R

- Required library: **coda**
  http://cran.r-project.org/web/packages/coda/coda.pdf
  - See WinBUGS example in R

Analysis using R

- Another option is using library: **boa**

  Bayesian **O**utput **A**nalysis

  http://www.public-health.uiowa.edu/boa/BOA.pdf

Run directly from R

- Required package for OpenBUGS: ***BRugs***
- Required package for WinBUGS: ***R2WinBUGS***
    - Write a Bugs model in a file with a ***.bug*** extension
    - Go into R
    - Prepare the inputs to the ***bugs*** function in R and run it
    - A WinBUGS window will pop up and R will freeze up
    - The model will now run in WinBUGS (Log window)
    - When WinBUGS is done, its window will close and R will work again.
    - See example in R

Useful links

- The BUGS Project
  http://www.mrc-bsu.cam.ac.uk/bugs/welcome.shtml
  http://www.mrc-bsu.cam.ac.uk/bugs/documentation/contents.shtml

- Woodworth, George G. *Biostatistics: A Bayesian Introduction. Introduction to WinBUGS. 2004*
  http://www.stat.uiowa.edu/~gwoodwor/BBIText/AppendixBWinbugs.pdf

- Sturtz S, Ligges U, Gelman A (2005). *R2WinBUGS: A Package for Running WinBUGS from R.* Journal of Statistical Software, 12(3), 1-16.
  http://cran.r-project.org/web/packages/R2WinBUGS/vignettes/R2WinBUGS.pdf

Useful links

- Plummer M, Best N, Cowles K and Vines K. (2006). *CODA: Convergence Diagnosis and Output Analysis for MCMC*. R News. 6(1), 7-11
  http://cran.r-project.org/doc/Rnews/Rnews_2006-1.pdf

- *Running WinBugs and OpenBugs from R*
  http://www.stat.columbia.edu/~gelman/bugsR/

Selected references

- Brooks SP and Gelman A (1998) *Alternative methods for monitoring convergence of iterative simulations*. Journal of Computational and Graphical Statistics. 7, 434-455.

- Lunn DJ, Thomas A, Best N, and Spiegelhalter D (2000) *WinBUGS -- a Bayesian modelling framework: concepts, structure, and extensibility*. Statistics and Computing, 10:325--337.

- Ntzoufras, I (2009) *Bayesian modeling using WinBUGS*. John Wiley and Sons.