LESSON 1 NOTES

USING RJAGS FOR BAYESIAN INFERENCE IN R: PROGRAMMING AND DATA ANALYSIS

**1. Course Content and Reading**

The course will show how to use the rjags library in R to develop a range of regression models, and how Bayesian inferences from the models can be obtained and represented/displayed in R. The focus in writing code will be on representing statistical models in JAGS. Use of R will be confined to calling inbuilt functions and routines, and data input; there will be no programming directly in R, apart from simple calculations.

Recommended reading is:

D Lunn et al, The BUGS Book - A Practical Introduction to Bayesian Analysis, CRC Press (2012).

This book is an excellent guide to BUGS, which is essential to using R interfaces to JAGS and BUGS. It is not specifically about R but any required instruction about R commands will be provided in the course materials.

**2 Introduction: Basic Principles of Bayesian Inference through MCMC Sampling**

2.1 Bayesian analysis combines a likelihood for the observed data y given unknowns θ, L(y|θ), and prior beliefs about densities for unknown parameters θ, π(θ), to provide updated inferences via the posterior density π(θ|y). For example in normal linear regression, the likelihood is normal, and priors are needed on unknown parameters (regression coefficients, residual variance). Note that there is no “correct” prior density, and quite a range of flexible choices are available in this regard. The posterior density can be written

π(θ|y)=k L(y|θ)π(θ),

where k is a constant. The posterior density can be maximised to estimate θ (maximum a posteriori estimation), in a similar way to maximum likelihood.

2.2 However, this is not always possible, and a richer range of inferences is obtained from sampling parameter values from the posterior density using Markov Chain Monte Carlo methods (MCMC). Such sampling typically involves many iterations (e.g. at least 5000 or 10000) to ensure coverage of the posterior density, partly because successive samples are dependent. MCMC sampling also typically involves multiple sampling sequences (known as chains) starting from different initial parameter values, in order to ensure convergence of the MCMC sampling.

2.3 So a typical MCMC run might set the number of iterations (say T) to T=10,000, and also set the number of chains (say C) to 2 or 3. Convergence is obtained when the chains have ‘forgotten’ their initial values, and the output from all chains is indistinguishable. Convergence may be assessed using Gelman-Rubin potential scale reduction factors (PSRF) available in R as MCMC diagnostics. The PSRF is an estimate of how much narrower the posterior for a parameter (or other model based unknown) might become with an infinite number of iterations. Values close to 1 (e.g. lower than 1.1) indicate convergence. A multivariate version of the PSRF is also available.

2.4 One can summarise the model (namely the assumed prior densities, likelihood assumptions, and resulting posterior density) as

θ ~ π(θ),

y ~ L(y|θ),

θ|y ∝ L(y|θ)π(θ).

Predictions can be obtained (Lunn et al, section 8.4) by sampling replicate data, as in

ypred ~ L(y|θ),

and such predictions can be viewed as extra unknown parameters. This representation is useful in considering JAGS/BUGS implementation (see below), which in fact just use the prior and likelihood details.

2.5 A particular sort of MCMC sampling is obtained when the full conditional density is a standard form (Normal, gamma, etc). The full conditional density for a particular parameter θj is the density conditional on both y and other parameters, denoted (say) as θ[-j]. If the full conditional is of standard form, we can update inferences by sampling directly from the analytical form (Gibbs sampling). Otherwise Metropolis or Metropolis-Hastings sampling is needed, which involves proposing candidate parameter values from a proposal density. While it is convenient, there is no particular reason to prefer Gibbs sampling or to restrict oneself to models that lead to Gibbs sampling.

2.6 Inferences from the MCMC sampling are contained in posterior summaries. Posterior summaries of parameter values (or other outputs) typically include posterior means or medians, and 95% credible intervals. These are statements that a parameter is located within the interval with a probability of 95%.

2.7 Additional forms of summarising inferences are available. For continuous parameters, kernel density plots can summarise the overall shape of the density (useful in showing departures from normality). For non-symmetric posterior densities a 95% highest probability density (or hpd) interval can also be used (any point inside the interval has higher density than any point outside it).

2.8 Other relevant outputs include probabilities on parameter intervals (e.g. the probability that parameter θj is positive), or probability comparisons between actual observations and model predictions. Thus a check whether values yi are under-predicted involves the indicators

Ci=I(yi<yi,pred)=I(yi,pred>yi).

where I(A)=1 if condition a applies, I(A)=0 otherwise. These checks can be included in the MCMC sampling, and their posterior means amount to probability estimates.

2.9 An important point to note: answers obtained in examples and exercises below may vary slightly due to stochastic variation during MCMC sampling.

**3 Bayesian Statistics using R**

3.1 The R platform is increasingly the major focus for shared statistical software and methods development. From the perspective of Bayesian data analysis, the advantages of the R environment include:

a) interfaces to existing general purpose and flexible Bayesian programming software originally developed for standalone use, such as BUGS and JAGS

b) access to an expanding range of packages within R for particular, sometimes relatively specialised, Bayesian applications

c) access to R graphical, data input and data export facilities.

3.2 It is possible to write code directly in R for MCMC routines to sample from the posterior density. However, this can be time consuming and subject to error. Packages such as JAGS and BUGS include a range of well-established MCMC sampling routines, so that the work in a Bayesian data analysis can be concentrated on choosing appropriate priors, and assessing different models and likelihood assumptions.

3.3 The current course focuses on the use of rjags in particular, though other interface routines (calling WINBUGS or OPENBUGS from R) are available. JAGS has coding advantages over BUGS, despite being very similar to it, and using the library rjags is advantageous in not involving a launching the standalone program. The rjags manual is available at

<http://cran.r-project.org/web/packages/rjags/rjags.pdf>

The rjags package can then be installed using:

install.packages("rjags"),

or using the packages/install packages menu option. This calls on the main JAGS package that can be installed from

<http://www.sourceforge.net/projects/mcmc-jags/files>

Note that the JAGS package will not be used in standalone form.

3.4 Each time subsequently, when starting an analysis in R, the rjags package (and any other packages used) needs to be invoked using the library or require commands, as in

require(rjags)

Also it may be advantageous to store input data files in a particular subdirectory, such as C://R Data, which can be made the working directory.

3.5 Except for small datasets, a simple option for inputting data is via text files, with data arranged in columns headed with variable names. So an rjags session might be started off with the following

require(rjags)

setwd("C://R Data")

D <- read.table("data.txt",header=T)

etc. Note that write.table can be used as a simple way of writing data from R, which might then be opened in EXCEL (with a space delimiter). One would simply use the command

write.table(D,"D.txt")

**4 Programming in JAGS (and BUGS)**

4.1 JAGS programs are very similar to those used in the WINBUGS and OPENBUGS programs, though there are some advantages with regard to code being economical that we mention below. A JAGS code is a statement of a statistical model, following the principle set out in 2.4 above, but only needing one to specify (a) the prior distributions π(θ) on parameters θ, and (b) the likelihood L(y|θ) assumed for the data.

4.2 The code may also stipulate model derived outputs (e.g. parameter comparisons), or contain indicators relevant to assessing hypotheses on parameters. Unknown stochastic quantities, both model parameters and model derived quantities, or hypothesis indicators, are collectively known as nodes.

4.3 JAGS and BUGS programs are declarative and do not operate sequentially. There is no prescribed order for programming statements of different kinds (e.g. it is not necessary that priors be specified before the likelihood). The program code is simply a description of the model (likelihood and priors), and of any additional stochastic nodes that may be monitored at each MCMC step. More detailed descriptions of the JAGS language along with lists of the available stochastic distributions and logical functions (e.g. absolute value of a quantity x) can be found in the JAGS manual available at

sourceforge.net/projects/mcmc-jags/files/Manuals/‎

More detailed descriptions of the BUGS language can be found in the OPENBUGS manual available at

http://mathstat.helsinki.fi/openbugs/ManualsFrames.html

4.4 Any JAGS or BUGS code is bracketed within a model statement:

model {text-based description of graph in JAGS or BUGS language}

Different bracket types are used for particular purposes. Round brackets () are used in in mathematical expressions, mathematical functions (e.g. logs or square roots), and in density statements. Curly brackets {} are used in for loops and to declare the start and end of the model. Square brackets [] are used to specify elements in vectors or arrays, both observations and parameters.

4.5 The coding principles in JAGS/BUGS draw on the principles used in R, such as for-loops for repeated calculation, indexing of vectors or matrices in square brackets, and in using lists as one way of inputting data. There are some important differences, such as in the specification of statistical densities. A major difference is that sampling the normal density in JAGS and BUGS (the dnorm function, see section 5) uses the inverse variance or precision, whereas the normal function in R uses the standard deviation.

4.6 The symbol ∼ is used in JAGS and BUGS to denote stochastic (probabilistic) relationships, for example that a parameter follows a particular density (see section 5). The left arrow <- symbol ('<' sign followed by '-' sign) denotes a deterministic relationship, as for example in defining a regression equation, or in data manipulations.

4.7 A for-loop is used in BUGS/JAGS for repeated calculations that apply to all elements in a vector or array. All calculated quantities or stochastic variables defined in a for-loop will need to be indexed, for example as in the following:

for (i in 1:n) {y[i] ~dnorm(mu[i],tau)

mu[i] <- beta[1]+beta[2]\*x[i]}

The first line is a stochastic relationship defining the normal density (see section 5) assumed for the observations y[i], while the second line is a deterministic relationship defining the regression means mu[i] in terms of parameters beta[1:2] and a predictor x[i].

4.8 Note that JAGS (but not BUGS) allows one to economically compress the above code as follows

for (i in 1:n) {y[i] ~dnorm(beta[1]+beta[2]\*x[i],tau)}.

That is, calculations are allowed in defining the parameters of densities. To set univariate normal priors on the beta parameters would require another for-loop, such as

for (j in 1:2) {beta[j] ~ dnorm(0,0.001)}

which specifies N(0,1000) priors.

**5 Available Densities**

5.1 A large range of densities are available in JAGS and BUGS. The following are most useful for the current course.

5.2 Normal density: y~dnorm(m,tau)

Normal with mean mu, variance 1/tau, precision (inverse variance) tau.

This density is useful both for specifying likelihoods (e.g. normal linear regression in Lesson 2), and for specifying priors on parameters that can have both positive and negative values (e.g. regression coefficients).

Constrained normal sampling can be used (e.g. a standard deviation can be assigned a normal prior constrained to positive values)

Normal sampling between limits (A,B):

y~dnorm(mu,tau) I(A,B)

Constrained normal sampling below B:

y~dnorm(mu,tau) I(,B)

Constrained normal sampling above A:

y~dnorm(mu,tau) I(A,)

This notation for constrained sampling applies to other densities.

5.3 Poisson density y ~ dpois(mu)

The Poisson density with mean mu, used as a likelihood for count data. If there is an exposure or "offset" E (e.g. time at risk, population at risk) then the underlying rate ν is obtained as μ=νE, which in JAGS can be coded

y ~ dpois(nu\*E).

5.4 Gamma and Related Densities

Gamma Density: y ~ dgamma(a,b), y>0

This refers to a Gamma with shape a, scale b, mean a/b, and variance a/b²,

p(y|a,b)= (ba/Γ(a)) ya-1exp(-by)

This is a density used for necessarily positive parameters, most commonly in setting priors, such as on precisions (inverse variances) or Poisson means.

Important Note: The inverse gamma for z=1/y and with parameters (a,b) is equivalent to a gamma for y with parameters (a,b). The inverse gamma has form

p(z|a,b)= (ba/Γ(a)) z-(a+1)exp(-b/z),

with E(z)=b/(a-1), var(z)=b/[(a-1)2(a-2)].

The inverse gamma is not however available directly in JAGS or BUGS.

Chi-square: y ~ dchisqr(k)

Same as gamma with a=k/2, b=0.5.

Exponential density: y ~ dexp(b)

Exponential with mean 1/b, p(y)=bexp(-by), namely a gamma with a=1.

5.5 Binomial and related densities

Binomial: y ~ dbin(p,n)

Binomial with n trials and probability of success p

Bernoulli density y ~ dbern(p)

Bernoulli with probability p. This is equivalent to a binomial in a sample of n=1, so can also be implemented using y~dbin(p,1)

Beta density: y ~ dbeta(a,b)

Beta with mean a/c, and variance ab/[c2(c+1)] where c=a+b. Usually used as a prior for a probability in a binomial or Bernoulli likelihood

5.6 Uniform density: y ~ dunif(a,b)

Uniform between a and b. Usually used only as a prior, for example a standard deviation can be assigned a uniform prior such as σ ~ U(0,100).

5.7 Student t density: y ~ dt(mu,tau,nu)

The Student t density with mean mu, nu degrees of freedom, and precision tau (i.e. variance 1/tau). Used as a heavier tailed alternative to the Normal, for data suspected to have outliers, or as a prior for regression coefficients.

5.8 Multivariate Normal density

y[1:K] ~ dmnorm(mu[1:K],Inv.Sigma[1:K,1:K])

Multivariate normal of dimension K, with inverse covariance matrix or precision matrix Σ⁻¹ denoted Inv.Sigma, and covariance Σ. The indexed multivariate Normal (e.g. bivariate regression) is demonstrated by

y[i,1:K]~dmnorm(mu[1:K],Inv.Sigma[1:K,1:K])

if only means for y1,..yK were specified in the model, or

y[i,1:K]~dmnorm(mu[i,1:K],Inv.Sigma[1:K,1:K])

if regressions were to be specified, such as mu[i,k] <- beta[k,1]+beta[k,2]\*x[i].

The multivariate normal is also used as a prior for regression coefficients.

5.9 Multinomial and Categorical Densities (category response)

y[1:K]~dmulti(p[1:K],n)

Multinomial with n trials, K categories (e.g. brand choices, mutually exclusive diagnoses) and probabilities (p₁,p₂,..,pK) with =1.

Categorical density

y~dcat(p[1:K])

This relates to a multinomial with n=1, and probabilities (p₁,p₂,..,pK), and generates a value between 1 and K.

Other densities: Weibull, Pareto, Double Exponential, Wishart, F, logistic.

**6 Logical Functions**

6.1 The following is a selection of useful logical functions widely needed in JAGS or BUGS programs. There are no “if statements”, but similar functionality is provided by step and equals statements:

S <- step(x) S=1 if x ≥ 0; S=0 otherwise

S <- step(x-b) S=1 if x ≥ b; S=0 otherwise

S <- step(x-b)-equals(x,b) S=1 if x >b; S=0 otherwise (i.e. strict inequality)

E <- equals(x1, x2) E=1 if x₁ = x₂; E=0 otherwise

These statements are equivalent to binary indicator functions I(A)=1 if condition A holds, I(A)=0 otherwise.

6.2 For vector manipulations/summaries the following are useful

sum(x[1:n]) Sum of the xi, Σxi

mean(x[1:n]) mean of elements (x₁,…,xn)

sd(x[1:n]) standard deviation of (x₁,…,xn)

inprod(u[], v[]) Σuivi

max.x <- max(x[]) maximum of array x[1,...,n] of length n

min.x <- min(x[]) minimum of array x[1,...,n] of length n

6.3 Other mathematical functions as follows:

abs(x) Absolute value, |x|

exp(x) Exponential transform of x

log(x) log to base e of x , namely loge(x) or ln(x).

logit(p) logit transform (used in binary data regression) log(p / (1 - p))

phi(x) standard normal cdf

pow(x, y) x to power y, xy

sqrt(x) square root of x, x0.5

round(x) nearest integer to x

trunc(e) greatest integer less than or equal to e

inverse(V[,]) V⁻¹ for symmetric positive-definite matrix V

**7. A Simple Model, with Example JAGS codes, and R listings**

7.1 We now consider an example of a widely applied conjugate model and how it might be coded in JAGS or BUGS. A conjugate model is obtained when the prior π(θ) and the posterior π(θ|y) follow the same density, except that the latter includes additional information from the data. Conjugate models are advantageous because the exact posterior distribution of the unknowns is known, and so one can easily simulate or derive quantities of interest analytically. However, for many quite simple models there is no conjugate prior available.

7.2 Suppose we have observations {y1,..,yn} (such as weights, heights, or blood pressure readings for a random sample of subjects), and assume a normal likelihood

yi ~ N(μ,σ2).

We wish to estimate the underlying population mean μ, and the precision τ, namely the inverse of the residual variance (the “residuals” here are yi-μ), so that τ=1/σ2. The higher the precision the more closely clustered are the observations around the mean. Under the conjugate prior we assume a gamma prior for τ, as is appropriate for a quantity which can only be positive,

τ ~ Ga(a,b),

or in JAGS code

tau ~ dgamma(a,b).

The numbers a and b are usually preset, namely taken as known or simply “known” in common terminology. For example we might take a=1, and b=0.01, and so we could write the prior in JAGS as

tau ~ dgamma(1,0.01).

7.3 We also assume a normal prior for μ, with pre-specified mean μ0, and variance V0, and to get a conjugate prior, we need to link the variance V0 to the sampled residual precision τ, in what is termed a conditional prior (see, for example, section 7.3 in the Cowles book, and section 5.3 in the Hoff reference above). In particular we can set V0=σ2/n0 where n0 is also pre-specified, and can be seen as a measure (usually at most 1) of prior sample size or prior “effective” sample size. So the conditional prior for μ is

π(μ|σ2)= N(μ0, V0) = N(μ0, σ2/n0).

For example, one might take n0=1 or n0=0.1. Remember that the Normal density in JAGS/BUGS involves the precision, namely

P0=1/V0=n0/σ2= n0τ,

so (assuming the setting μ0=0) the JAGS code might be

mu ~ dnorm(0, n0\*tau).

7.4 Suppose there are n=10 observations, and we take μ0=0, n0=0.1, and τ ~ Ga(1,0.01). Then we can put together a corresponding JAGS code as

model {for (i in 1:10) {y[i] ~ dnorm(mu,tau)}

tau ~ dgamma(1,0.01)

mu ~ dnorm(0, n0\*tau)

sigma2 <- 1/tau

n0 <- 0.1}

Note that we could also set n0=0.1 in the R commands preceding the model.

7.5 The first line in the above code is an example of a for-loop. The second and third lines denote stochastic relationships, and the last two lines are deterministic relationships. The command

sigma2 <- 1/tau

exemplifies how we can monitor a derived parameter, namely the residual variance (and make posterior inferences about it). This is so even though the prior is on τ=1/σ2.

7.6 The JAGS program can either be embedded within the sequence of R commands using a cat command. The central rjags model command is the jags. model command, exemplified by

M <- jags.model("model.jag", inits = INI, n.chains = 2, n.adapt=1000)

The first argument model.jag is the name of the file containing the JAGS code, either listed earlier using a cat command or saved into the working directory;

Inits is a list of lists of initial values (one for each chain);

n.chains specifies the number of parallel chains;

n.adapt specifies the number of iterations assumed for an initial sampling phase during which MCMC sampling routines adapt their behaviour to maximize efficiency (e.g. a Metropolis-Hastings algorithm may need to calibrate its proposal density). This is typically not relevant to simple models, conjugate examples, but included for illustrative purposes.

7.7 Suitable initial values for positive parameters (precisions, standard deviations) are values such as 1, 5, 0.5, 0.1, etc. Suitable starting values for normal means, regression coefficients (and other parameters with both positive and negative possible values) are zeroes, with perturbation to ensure different initial values in each chain, such as

beta=c(0,0)

in one chain, and

beta=(0.5,0.5)

in another. Choice of initial settings may also depend on the application, and on using other suitable knowledge (e.g. results from classical fitting). If a parameter has a known value (e.g. the first regression parameter is set to zero as a corner constraint when a predictor is categorical) then the initial value is specified as NA.

7.8 Unless input directly to the R Console, relevant data can be input and declared using assign commands. A useful working practice is to store data files in a subfolder such as C://R data or C://R files. Then use the working directory setwd command to read in data from that subdirectory.

For example, if a linear regression model involves data arrays y, x1 and x2, then these can be read in using a read.table command and then the arrays specified as follows:

# columns headed y,x1,x2

setwd("C://R files")

D <- read.table("data.txt",header=T)

y <- D$y; x1 <- D$x1; x2 <- D$x2

The variables y, x1, x2 can then be referred to in a subsequent JAGS code. Instead of the assign commands (y <- D$y etc), an alternative is the attach command. So one has

setwd("C://R files")

D <- read.table("data.txt",header=T)

attach(D)

7.9 Having inputted the data, and defined the model (via the code in 7.4, and the jags.model assignment in 7.6), we wish to define the number of samples to be drawn and also the parameters (or other nodes) to monitor. This involves the coda.samples command, exemplified by

R <- coda.samples(M, c("mu","sigma2"), n.iter=10000)

saying that parameters mu and sigma2 are to be monitored. A summary of R using the command

summary(R)

will then produce posterior mean and standard deviation for each monitored parameter, plus the estimated standard error of the mean. Also produced are quantile summaries (posterior 2.5%, 25%, 75%, 97.5% points and posterior median). A 95% credible interval is then provided by the posterior 2.5% and 97.5% points.

Other quantiles can be obtained using the jags.samples command. For example to obtain 10%, 20%, etc quantiles for mu (using 1000 samples), one would specify

J <- jags.samples(M, c("mu"),1000)

quantile(J$mu, seq(0, 1, by=.1))

Traceplots can be obtained using traceplot(R), though the command

plot(R)

also provides trace plots, as well as kernel density estimates for all monitored parameters.

7.10 Returning to the normal data example in 7.2-7.5, assume first of all direct data input to the R Console. Then a full sequence in R, with JAGS code embedded, is exemplified by

library(rjags)

data <- list(y=c(10,9,9,8,9.5,7,12,11,8,10.5))

# JAGS CODE

cat("data {n0 <- 0.1}

model { for (i in 1:10) {y[i] ~ dnorm(mu,tau)}

tau ~ dgamma(1,0.01); sigma2 <- 1/tau

mu ~ dnorm(0, n0\*tau)}

", file="normaldata.jag")

INI <- list(list(mu=10,tau=1),list(mu=9, tau=0.1))

M <- jags.model("normaldata.jag",data=data,inits=INI,n.chains=2,n.adapt=500)

R <- coda.samples(M,c("mu","sigma2"),n.iter=25000)

summary(R); traceplot(R)

where data {n0 <- 0.1} exemplifies a JAGS “data statement”. An equivalent code is

library(rjags)

data <- list(y=c(10,9,9,8,9.5,7,12,11,8,10.5),n0=0.1)

# JAGS CODE

cat("model { for (i in 1:10) {y[i] ~ dnorm(mu,tau)}

tau ~ dgamma(1,0.01); sigma2 <- 1/tau

mu ~ dnorm(0, n0\*tau)}

", file="normaldata.jag")

INI <- list(list(mu=10,tau=1),list(mu=9, tau=0.1))

M <- jags.model("normaldata.jag",data=data,inits=INI,n.chains=2,n.adapt=500)

R <- coda.samples(M,c("mu","sigma2"),n.iter=25000)

summary(R); traceplot(R)

It will be seen that initial values of 9 and 10 have been adopted for the normal mean, mu. It is suggested to experiment with less obviously data based choices.

7.11 For larger data sets, it is often preferable to read in data from an external file. A recommended data input option is for text files with columns headed by variable names, though a range of other input options are possible. In the present example, suppose we created a data file called "normaldata.txt", and saved it to C://R files. This data file would have the form

y

10

9

9

….

10.5

7.12 The above call sequence becomes

library(rjags)

setwd("C://R files")

D <- read.table("normaldata.txt",header=T)

# JAGS CODE

cat("data {n0 <- 0.1}

model { for (i in 1:10) {y[i] ~ dnorm(mu,tau)}

tau ~ dgamma(1,0.01); sigma2 <- 1/tau

mu ~ dnorm(0, n0\*tau)}

", file="normaldata.jag")

INI <- list(list(mu=10,tau=1),list(mu=9, tau=0.1))

M <- jags.model("normaldata.jag",data=D,inits=INI,n.chains=2,n.adapt=500)

R <- coda.samples(M,c("mu","sigma2"),n.iter=25000)

summary(R); traceplot(R)

**8 Additional MCMC Diagnostics and Parameter Sumamaries.**

8.1 One way to see if the sampling has converged is to see how well the chain is mixing, or moving around the parameter space. If the chain is taking a long time to move around the parameter space, then it will take longer to converge.

Trace plots show sampled parameter values against the iteration number, and will show whether the chain gets stuck in certain areas of the parameter space. The commands

plot(R)

or

traceplot(R)

provide trace plots.

8.2 Highest posterior density intervals for the parameters can be obtained with the command

HPDinterval(R, prob = 0.95).

8.3 The additional commands

gelman.diag(R)

gelman.plot(R)

provide the actual values of the PSRF (see item 2.3), and the evolution of the PSRF as the number of iterations increases.

8.4 Summary measures of fit can also be obtained. These are penalized fit measures, with a penalty measure (representing model complexity) adjusting the overall fit (the expected deviance). The DIC (Spiegelhalter et al 2002) is calculated by adding the “effective number of parameters” (pD) to the expected deviance. In rjags, the definition of pD used by dic.samples is the one proposed by Plummer (2002). Thus one would specify something like

dic.samples(M, n.iter=1000, type="pD")

where M has been specified in the jags.model command.

**9. Questions**

1 Include a statement in the JAGS code segment in 7.10 to monitor the residual standard deviation σ (denoting it sigma in the code), and also modify the coda.samples command to monitor sigma additionally. Otherwise use the same data input method, and other settings as in the listing in 7.10. Is the lower point of the 95% credible interval for σ (a) below 2 or (b) above 2.

2. Include a statement in the JAGS code segment in 7.10 to obtain the posterior probability that the residual standard deviation σ exceeds 2.5. Note that if the result of an indicator function calculation, such as

ind.mu <- step(mu-9)

is followed over the MCMC sampling, the posterior mean of the indicator function is a posterior probability. Is the required posterior probability (a) below 0.02 or (b) above 0.02.

3 Include a statement in the JAGS code segment to monitor the coefficient of variation σ/μ, denoted in the code as CV, and also modify the coda.samples command to monitor CV additionally. Otherwise use the same data input method, and other settings as in the listing in 7.10. Is the upper point of the 95% credible interval for σ/μ (a) below 0.25 or (b) above 0.25.

4. Include a statement in the JAGS code segment to obtain the posterior probability that the coefficient of variation σ/μ exceeds 0.2. Is the required posterior probability (a) below 0.3 or (b) above 0.3.

5. In 7.10, modify the data input, so that observation 1 is y1=1 instead of y1=10. Also include in the code an extra command to sample replicate data (model predictions). These are sampled from exactly the same model used to fit the data. Thus

for (i in 1:10) {y[i] ~ dnorm(mu,tau)

yrep[i] ~ dnorm(mu,tau)

….}

Additionally include an indicator function (using the step command) to check whether yrep[i] exceeds y[i], namely whether the model prediction exceeds the actual observation. Find the posterior probabilities that yrep[i] exceeds y[i], namely

Pr(yrep,i>yi|y).

Questions

5.1) For which observation is the probability Pr(yrep,i>yi|y) highest

5.2) Is the probability in Q5.1 (a) under 0.95, or (b) over 0.95

5.3) Does this model check indicate (a) a successful, or (b) a problematic model?

6. The t-distribution is symmetric and bell-shaped, like the normal distribution, but has heavier tails, meaning that it is more prone to producing values that fall far from the overall mean μ. Keep y1=1 as in Q5, but modify the likelihood assumption in Q5 to a Student t density with 4 degrees of freedom. Thus the likelihood statement is now

y[i] ~ dt(mu,tau,4).

Questions

6.1) For which observation is the probability Pr(yrep,i>yi|y) highest

6.2) Is the probability in Q6.1 (a) under 0.95, or (b) over 0.95

6.3) Is the posterior mean for μ under the Student t model (a) higher or (b) lower than under the Normal likelihood model.

**10. References**

Spiegelhalter, D et al (2002), Bayesian measures of model complexity and fit Journal of the Royal Statistical Society Series B 64, 583-639.

Plummer, M. (2002) Discussion of the paper by Spiegelhalter et al. Journal of the Royal Statistical Society Series B 64, 620.