8 WinBUGS/OpenBUGS

$\boxed{\text{WinBUGS = \textbf{B}ayesian inference \textbf{U}sing \textbf{G}ibbs \textbf{S}ampling}}$

WinBUGS is a computer program aimed at making MCMC available to applied researchers. Its interface is fairly easy to use, and it can also be called from programs such as R. WinBUGS is free and can be found on the website:

http://www.mrc-bsu.cam.ac.uk/bugs/winbugs/contents.shtml

OpenBUGS program, found at: http://www.openbugs.info/w/

Given a likelihood and prior distribution, the aim of both WinBUGS and OpenBUGS is to sample model parameters (and other unknown quantities) from their posterior distribution. After the parameters have been sampled for many iterations, parameter estimates can be obtained and inferences can be made by using the sample as approximation of the posterior distribution.

For a given application project, three files are used:

1. A program file containing the model specification.
2. A data file containing the data in a specific (slightly strange) format.
3. A file containing starting values ('initials') for model parameters (optional).

File 3 is optional because WinBUGS/OpenBUGS can generate its own starting values. There is no guarantee that the generated starting values are good starting values, though. All three files can be written in one if manually choosing by click-and-point the model code, data and inits.

Advice for new users:

1. Step through the simple worked example in the tutorial.
2. Try other examples provided with this release
(see Examples Volume 1 and 2, also Vol 3 in OpenBUGS)
3. Edit the BUGS language to fit an example of your own.

It is easiest to take existing code for a simple model and modify that for your purpose. It as been recommended that 'users should already be aware of the background to bayesian Markov chain Monte Carlo methods'. That's why it was included in the introduction part.

The current Metropolis MCMC algorithm is based on a symmetric normal proposal distribution, whose standard deviation is tuned over the first 4000 iterations in order to get an acceptance rate of between 20% and 40%. All summary statistics for the model will ignore information from this adapting phase. In OpenBUGS, the samplers have been further developed and this process is expected to continue. WinBUGS will no longer be updated. Hence, version 1.4.3 will be the last of WinBUGS.

Strong recommendation: the first step in any analysis should be the **construction of a directed graphical model**. Briefly, this represents all quantities as nodes in a **Directed Acyclic Graph**

(DAG), in which arrows run into nodes from their direct influences (parents). The model represents the assumption that, given its parent nodes pa[v], each node v is independent of all other nodes in the graph except descendants of v, where descendant has the obvious definition. This visualization of the model is very useful for presenting the model in a glance.

Nodes in the graph are of three types.

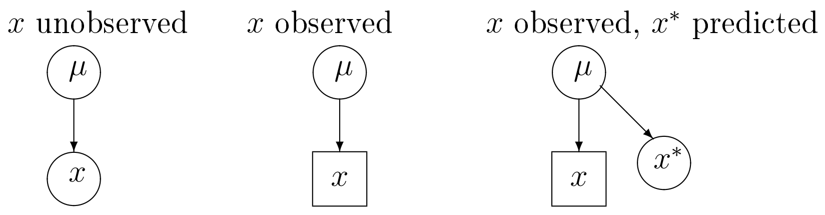
1. **Constants** are fixed by the design of the study: they are always founder nodes (i.e. do not have parents), and are denoted as rectangles in the graph. They must be specified in a data file.

2. **Stochastic nodes** are variables that are given a conditional distribution, and are denoted as ellipses in the graph; they may be parents or children (or both). Stochastic nodes may be observed in which case they are data, or may be unobserved and hence be parameters, which may be unknown quantities underlying a model, observations on an individual case that are unobserved say due to censoring, or simply missing data. They are coded with $\sim$ before the specified conditional distribution. (e.g. `x` $\sim$ `dnorm(mu,tau)`).

3. **Deterministic nodes** are logical functions of other nodes. Note that they are not allowed to be given data values. Data should always be given to a stochastic node as an observed value for that. Therefore, we cannot specify a structure where e.g. $X \sim \mathrm{N}(\mu, \tau)$ and $Y \sim \mathrm{N}(\mu, \tau)$ and $Z \leftarrow (X+Y)/2$, and then assign Z some observed data value. This would lead to an error message indicating multiple definition of Z. Instead, we need to define Z as a stochastic node $Z \sim \mathrm{N}(\mu, 2\tau)$, to be able to assign data value for it. This has been causing some headache when trying to define distributions implicitly. It is not possible. A conditional distribution needs to be chosen for every stochastic node in the graph. Deterministic nodes are coded with $\leftarrow$. (e.g. `x <- log(z*z)/2 + u`).

Stochastic quantities can be specified as data by giving them values in a data file, in which values for constants are also given.

Example: $x \sim \mathrm{N}(\mu, 1)$ with prior $\mu \sim \mathrm{N}(0, 10^4)$. The DAG would run from stochastic node μ (parent of x) to stochastic node x (child of μ). The latter would be assigned some data value, which leaves only μ as an unknown parameter for which the posterior will be computed, given data value for x. A deterministic node could be added by defining e.g. $logmu \leftarrow \log(\mu)$, for monitoring the MCMC samples for the log of μ, if its posterior happens to be of any interest. The constant value of 1 for the variance in the conditional model of x could be given as a fixed parameter τ which then needs to be given also as data, $\tau = 1$. This would be a founder node in the DAG since there would be no parents for it. To summarize: these specifications together are needed for constructing the MCMC sampler (inside WinBUGS/OpenBUGS) for computing the posterior $\pi(\mu \mid x)$. If x is not given a data value, it too remains an unknown stochastic node. Effectively, it would then be simulated from the prior predictive distribution, whereas μ would be simulated from its prior only. After observing x, the posterior predictive distribution $\pi(x^* \mid x)$ for a new, x^*, observation can be computed simply by adding x^* as an unobserved node in the graph, with the same conditional distribution as there was for x, given μ.



8.1 Steps of installing WinBUGS/OpenBUGS

Go to the website (either Win- or Open-) and follow instructions - it usually works. Because the development of WinBUGS is not to be continued, its compatibility with other new software in the future is not sure. New updates will appear for OpenBUGS. If installing WinBUGS, you should get version 1.4 which is then upgraded to 1.4.3 by installing a patch as instructed. Also, a keyfile was required for getting the fully functional version. This keyfile used to expire at the end of the year, and new keyfiles were mailed to registered users. However, finally an 'immortal' key was given for all users, now from the Website. In OpenBUGS these steps are not involved. You just install the latest version of OpenBUGS.

Installation in Windows machines has usually been straightforward. Some difficulties(?) may occur with Linux/Mac, but it is possible to run WinBUGS from Mac and OpenBUGS from Linux (and also WinBUGS via emulators). For detailed instructions with each platform, you should carefully read the installation instructions provided in the websites. For example, they say:

Note: There appears to be a problem with installing WinBUGS and/or various patches in Windows Vista. Vista doesn't seem to like anyone overwriting files in the "C:\Program Files" directory (regardless of permissions). Hence we recommend that WinBUGS be installed elsewhere, e.g. "C:\".

If all else fails (for example with a 64-bit machine), you can download a zipped version of the whole file structure and unzip it into Program Files or wherever you want it. WinBUGS makes no changes to the Registry.

I have also installed WinBUGS on a memory stick. Seems to be running!

8.2 Steps of running WinBUGS/OpenBUGS models

Assume you have an existing BUGS model code in a file ('.odc' or '.txt'). Assume also that the data list is included in the same file (a list written below the model code).

1. Open the file in WinBUGS/OpenBUGS
2. Open `Model > Specification...`
3. Check the model's syntax
4. Load data
5. Compile model
6. Set initial values (from a preset list, or let the software generate them)
7. Run the model `Model > Update...`

8. After convergence, set parameters of interest for monitoring. `Inference > Samples...`
9. Run again the model to get sufficiently large sample
10. See the output graphically (`history, density`), and summary statistics (`stats`)

8.3 Structure of the model

The syntax and form of a model in WinBUGS follows (nearly) directly from the structure of the required densities in the Bayes formula. In OpenBUGS, the structure of the language is the same, with some added new functions or distributions which gradually may evolve. Other features have also emerged in OpenBUGS (see the Webpage for details), including the possibility to get your model code printed with LaTeX commands. However the core of the model specification language is still the same. Understanding of the product rule and bayes formula, as well as other basic theorems of probability calculus is as essential as understanding grammatical rules and structure of sentences in natural languages. We need to specify a conditional distribution of data, and a prior. These can consist of several conditional distributions. The whole structure is convenient to draw as a Directed Acyclic Graph (DAG) which you can frequently find in BUGS examples. (There are some conventions to draw different arrows for stochastic dependencies and deterministic dependencies, etc.). This makes a visual expression of the **logical structure** for a complete specification of a joint probability model. It is this logical structure we need to code for WinBUGS/OpenBUGS.

The joint posterior density is always fully specified when all these necessary parts are defined. Therefore, WinBUGS is a **declarative** language, as opposed to **procedural** programming languages. (**This is important to remember**). In a procedural language the following code could be valid:

```
X := 1;
Y := 1;
Z := X+Y;
```

but the following would not compute procedurally:

```
Z := X+Y;
X := 1;
Y := 1;
```

In WinBUGS/OpenBUGS, the order of these statements would not matter, because only the logical structure is defined which can be *written out* in any order, as long as all the quantities are defined somewhere and their combination defines a valid model.

In WB, we define a chain of conditional distributions that was obtained from the product rule when writing the Bayes formula. Each variable $v \in V$ in the model can be a 'child node' that is conditionally dependent on its 'parent nodes':

$$\pi(V) = \prod_{v \in V} \pi(v \mid \text{parents}\{v\}),$$

and the last variables in this chain have no further parents, i.e. their conditional distribution does not depend on any further variables - it is the prior distribution. The whole structure specifies a bayesian model. A simple example (assuming data x, y, n, m) could be:

```
model{
x ~ dbin(px,n)
y ~ dbin(py,m)
px ~ dbeta(a,b)
py ~ dbeta(a,b)
a ~ dexp(1)
b ~ dexp(1)
}
```

or a linear model (assuming data x, y):

```
model{
for(i in 1:N){
 y[i] ~ dnorm(mu[i],tau)
 # note that: tau = 1/sigma^2
 mu[i] <- alpha + beta * (x[i]-mean(x[]))
 }
alpha ~ dnorm(0,0.0001); beta ~ dnorm(0,0.0001)
tau ~ dgamma(0.001,0.001)
}
list(N=5,x=c(1,2,3,4,5),y=c(1,3,3,3,5))
```

Comment lines can be written, starting with '#', and sufficient commenting is indeed recommended! In the binomial model above the DAG would be:

$$\begin{tikzcd}
& \alpha, \beta \arrow[dl] \arrow[dr] & \\
p_x \arrow[d] & & p_y \arrow[d] \\
x & & y
\end{tikzcd}$$

A DAG is really the skeleton that gives the necessary structure for a valid bayesian model as well as for a valid WB model code. (The exception is that in WB we can also define Gibbs sampling algorithm directly using 'full conditionals').

Since cycles are not allowed in a DAG, then how to define models where some variable has some feedback into itself? For example, the size of a population drives population growth which again determines the population size. The question is: what is the probability model for this? It is a model of a stochastic process. The variables need to be indexed with respect to time, so that the conditional distribution of X_{t+1} depends on X_t, and this can be written as a DAG without cycles. Alternatively, (but this can be more complicated), we could try to solve the conditional probability distribution for the whole set of values $X_1, \ldots, X_t$, given some other parameters of the model. But if the X variables are unknown, their simulation might require block updating which is not possible in WinBUGS which is based on single site updating algorithms (unless there are extensions available). In General, Gibbs sampling theory allows block updating if we can just solve what the full conditional density for a block (vector of parameters) is.

doodle BUGS
Models can be defined in WB either by writing the corresponding WB code, or by drawing the DAG using doodle-BUGS. Once the 'doodle' is defined, the corresponding WB code is automatically generated. But the opposite is not possible: a picture of a DAG cannot be generated from winBUGS code, you need to draw it elsewhere. But the WB language is much more versatile than doodle-BUGS, so it is best to learn to write WB codes, and do drawing of DAGs elsewhere.

8.3.1 Example: different codes, same model

```
p ~ dbeta(1,1)
x ~ dbin(p,n)
```

```
p ~ dbeta(a,b)
a <- x+1 ; b <- n-x+1
```

```
p ~ dunif(0,1)
pr <- exp(logfact(n)-logfact(x)-logfact(n-x)+x*log(p)+(n-x)*log(1-p))
one ~ dbern(pr); one <- 1
```

```
z ~ dnorm(0,1)
p <- phi(z)
x ~ dbin(p,n)
```

```
p ~ dunif(0,1)
for(i in 1:x){
result[i] ~ dbern(p); result[i] <- 1
}
for(j in x+1:n){
result[j] ~ dbern(p); result[j] <- 0
}
```

8.3.2 Posterior 'queries'

Once we can simulate from the posterior with WB, we can derive answers to several questions, or 'queries', that depend on the unknown parameters described by the posterior. For example, in comparing two populations with unknown prevalences p_1 and p_2, we might be interested in the following probabilities:

$$P(p_1 > p_2 \mid \text{data}), \quad P(|p_1 - p_2| > c \mid \text{data}), \quad P(p_1/p_2 > 1 \mid \text{data}), \quad P(p_1^2 + p_2^2 > c \mid \text{data})$$

which could correspond to some hypothesis. The last example could be related to e.g. genetical application in which p_i would be the prevalence of allele i in Hardy-Weinberg -equilibrium. The classical statistical approach of hypothesis testing concerning parameters and their transformations is replaced in bayesian context by computation of these probabilities. For example, the posterior density of risk ratio p_1/p_2 could be visualized as the simulated empirical distribution from WB, but we could also compute $P(p_1/p_2 > 1 \mid \text{data})$ as an answer to the question if the risk ratio is larger than one. This could be done by the following code, using `step`-function:

```
model{
for (i in 1:2){
x[i] ~ dbin(p[i],n[i])
p[i] ~ dbeta(1,1)
}
Pabsdiff <- step( abs(p[1]-p[2])-c)
}
```

Here, `Pabsdiff` is an indicator variable (remember from the preliminaries). When we compute the average of this indicator over the MCMC simulation $i = 1, \ldots, N$ we get an approximation of the required probability:

$$\frac{1}{N}\sum_{i=1}^{N} I_i \approx E(I) = 1 \times P(I=1) + 0 \times P(I=0) = P(I=1)$$

Asking for prediction:

We might also need a prediction for a forthcoming variable x_{new} in some forthcoming sample of size m_1, under unknown prevalence p_1. This could be computed simply by adding the following:

```
xnew[1] ~ dbin(p[1],m[1])
```

In the example of linear regression, we could ask for a prediction of y_6 at the next point x_6. This could be achieved by adding one more step in the loop by setting $N + 1$ instead of N, and by adding `NA` in the data list in place of `y[6]`.

8.4 Data structures

Anything that is not an unknown (random) quantity in the model, has to be fixed value, i.e. given as data or constant. Data are listed separately from the model code, for example:

```
list(x=4,
     y=c(3.5,7.2,9.1),
     z=structure(
        .Data=c(7,3,5,1,8,2),
        .Dim=c(2,3)))
```

which defines a scalar x, vector y and a matrix z of size 2×3. Data matrices can also be defined in this form:

```
 z[,1] z[,2] z[,3]
 7     3     5
 1     8     2
 END
```

so that first index of z needs to be left empty, and there must be an empty line after `END`. You can avoid much trouble if you always check carefully that you have indexed your data correctly. There are no useful tools for checking data inconsistencies within WinBUGS. When data variables have been defined and assigned, there should be a conditional distribution for them in the code. For example, if variable y is given as data, then we might have a model directly for it:

```
y ~ dnorm(mu,tau)
```

Alternatively, we might be interested in modeling some transformation of this variable, which could be done as:

```
 yy <- log(y)
 yy ~ dnorm(mu,tau)
```

Of course, we might have calculated the transformed y already beforehand, and then use that as data. Note that the previous use of transformations within the code is actually against WB syntax which prohibits multiple definitions of the same node. This is the exception to the rule. Otherwise, you get error messages: 'multiple definition of yy'. An error would be caused also if variable y was a vector with values given in the data, and some values would be missing ('`NA`'). The missing values would then be stochastic nodes and the above construction would lead to error.

8.5 Data from R

If data are in some other format in some other software, you have to find out how to make it in WB format. Note: in R, the syntax for `structure` is similar, but the indexing of rows and columns has opposite order. There are some tools that can be used for converting data:

http://www.mrc-bsu.cam.ac.uk/bugs/weblinks/webresource.shtml

For example, for Matlab, some tools are also at:

http://www.cs.ubc.ca/ murphyk/Software/MATBUGS/matbugs.html

See BMUW, p. 103, and also WinBUGS manual 'Model Specification' > 'Formatting of data': assume the following matrix x (of size 2*3) is the original object. In R, this could be defined by `x<-matrix(c(1,2,3,4,5,6),2,3,byrow=TRUE)`:

```
      [,1]   [,2]  [,3]
 [1,]   1       2     3
 [2,]   4       5     6
```

To get this in the format of 'list' for WinBUGS, we could produce almost similar printing in R, by using commands:

```
  x2 <- list(x=t(x))
  dput(x2)
```

Alternatively, to print in a file: `dput(x2,'filename.txt')`. Either way, this should print out something like:

```
 structure(list(x=structure(c(1,2,3,4,5,6),
                .Dim=c(3,2))), .Names="x")
```

This print out would be suitable for copying to BUGS, but some corrections need to be edited: (1) remove the 'structure(' from the beginning, (2) remove the ',.Names="x")', and (3) reverse the dimensions. After this editing, we have

```
 list(x=structure(.Data=c(1,2,3,4,5,6),.Dim=c(2,3)))
```

And this corresponds to the original object we wanted. The most important thing is to reverse the order of the dimension argument. For example:

```
 x <- array(c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18),dim=c(3,3,2))
```

produces this in R:

```
, , 1

      [,1] [,2] [,3]
 [1,]    1    4    7
 [2,]    2    5    8
 [3,]    3    6    9

, , 2

      [,1] [,2] [,3]
 [1,]   10   13   16
 [2,]   11   14   17
 [3,]   12   15   18
```

To reverse the indexing in R, use:

```
dimx <- dim(x) xx <- array(dim=rev(dimx)) for(i in 1:dimx[1]){
  for(j in 1:dimx[2]){
    for(k in 1:dimx[3]){
      xx[k,j,i]<-x[i,j,k]}}}
```

which will produce the following

```
, , 1

      [,1] [,2] [,3]
 [1,]    1    4    7
 [2,]   10   13   16

, , 2

      [,1] [,2] [,3]
 [1,]    2    5    8
 [2,]   11   14   17

, , 3

      [,1] [,2] [,3]
 [1,]    3    6    9
 [2,]   12   15   18
```

Finally, use `dput(xx)` to print, and then modify the output as before before importing to WinBUGS:

```
list( x= structure(.Data=c(1, 10, 4, 13, 7, 16, 2, 11, 5, 14, 8, 17,
3, 12, 6, 15, 9, 18), .Dim = c(3, 3, 2)))
```

After you have imported the data in WinBUGS, you can (after compiling the model) view it by selecting `Info>Node Info>Values`. The result would be:

```
 x[1,1,1]      1.0
 x[1,1,2]      10.0
 x[1,2,1]      4.0
 x[1,2,2]      13.0
 x[1,3,1]      7.0
 x[1,3,2]      16.0
 x[2,1,1]      2.0
 x[2,1,2]      11.0
 x[2,2,1]      5.0
 x[2,2,2]      14.0
 x[2,3,1]      8.0
 x[2,3,2]      17.0
 x[3,1,1]      3.0
 x[3,1,2]      12.0
 x[3,2,1]      6.0
 x[3,2,2]      15.0
 x[3,3,1]      9.0
 x[3,3,2]      18.0
```

8.6 Indexing and looping

Indexing of variables is an efficient way of simplifying and shortening WB code. You just need to be careful in that the indexing is correct. For example:

```
for(i in 1:3){ y[i] ~ dnorm(mu,tau)  }
```

It is also possible to define a distribution of a vector variable:

```
y[1:3] ~ dmnorm(mu[1:3],tau[1:3,1:3])
```

Here, it is sufficient to specify the indexes on the left hand side for y, whereas μ and τ could be written in the form `mu[]` and `tau[,]` which would include the whole vector or matrix. Also, sums could be defined over the whole vector:

```
 s <- sum(theta[])
 s2 <- sum(eta[,1])
```

Remember, a loop is just a way of writing repeated similar expressions in short. It is still just a collection of logical statements assembled together. In setting the limits for a loop we can either write it `for(i in 1:3)`, or specify the limit(s) as a constant with data: `for(i in 1:L)`, `list(L=3)`. If L is not given in data, we get the message: 'variable L is not defined'. For easier editing of new model versions, all constants should be defined together with data - in a single place. The model code becomes more versatile then. It is also convenient to use nested looping:

```
 for(i in 1:I){
    for(j in 1:J){
    y[i,j] ~ dpois(mu[i,j])
    log(mu[i,j]) <- mu0 + mu1*x1[i] + mu2*x2[j] + e[i,j]
    }
 }
```

The most frequent coding errors with loops are forgetting the braces, "}", or wrong position of the braces, or forgetting the indexing, or wrong indexing. Often this results to an error message: "multiple definition of..."

The range of indexing cannot be random. The following approach might be useful in such case:

```
for(i in 1:N){
 ind[i] <- 1 + step(i-K - 0.01)
 y[i] ~ dnorm(mu[ind[i]],1)
}
```

Here, variable K is unknown, K $\in \{1, \ldots, \mathrm{N}\}$, and it controls which variables y_i are modelled as $N(\mu_1, 1)$. (Can be useful in changepoint-problems).

8.6.1 Example: change point estimation

Assume data about monthly accidents y_i. This can be modeled using Poisson distribution, with parameter μ_1 before the change point and μ_2 after the change point. Prior distribution for the location of the change point could be discrete uniform $1/N$, where N is the number of months. The model assumes that μ_1 applies at least to the first month, and that it could also apply to the last month, in which case there would not be a change point at all.

```
 model{
 for(i in 1:N){
 y[i] ~ dpois(mu[ind[i]])
 ind[i] <- 1 +step(i-K-0.01)
 pk[i] <- 1/N
 }
 K ~ dcat(pk[]); mu[1] ~ dgamma(0.01,0.01); mu[2] ~ dgamma(0.01,0.01)
 }
```

An example with real data: coal mining accidents in Britain 1851-1962. (Carlin et al: Hierarchical Bayesian Analysis of Changepoint Problems. Appl. Statist. (1992). 41, No. 2, pp.389-405). Did improvement in technology and safety practices have an actual effect of the rate of serious accidents? When did the change actually occur? This could be modeled using the following code.

```
model{
 for(year in 1:N){
 T[year] <- year+1850
 D[year]~dpois(mu[year])
 log(mu[year])<-b[1]+step(year-changeyear)*b[2]
 }
 for(j in 1:2){b[j]~dnorm(0,0.0001)}
 actual<-changeyear+1850
 changeyear ~ dunif(1,N)
 mu1 <- exp(b[1])
 mu2 <- exp(b[1]+b[2])
}
 list(D=c(4, 5, 4, 1, 0, 4, 3, 4, 0, 6,
      3, 3, 4, 0, 2, 6, 3, 3, 5, 4,
      5, 3, 1, 4, 4, 1, 5, 5, 3, 4,
      2, 5, 2, 2, 3, 4, 2, 1, 3, 2,
      1, 1, 1, 1, 1, 3, 0, 0, 1, 0,
      1, 1, 0, 0, 3, 1, 0, 3, 2, 2,
      0, 1, 1, 1, 0, 1, 0, 1, 0, 0,
      0, 2, 1, 0, 0, 0, 1, 1, 0, 2,
      2, 3, 1, 1, 2, 1, 1, 1, 1, 2,
      4, 2, 0, 0, 0, 1, 4, 0, 0, 0,
      1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
      0, 0),N=112)
```

Some initial values might be needed to get this running:

```
list(b=c(0,0),changeyear=50)
```

8.7 Logical expressions

Long expressions can be too long for WinBUGS. You would then get the error message: "logical expression too complex". This can be avoided only by using additional variables:

```
a <- g + t + g*u + 7*pow(w,s)
b <- r + sqrt(h) - inprod(z[],zz[]) + e/p
c <- a+b
```

instead of writing the whole expression as a one-liner. Some useful logical functions in WinBUGS are (there are more in OpenBUGS):

```
abs(e)
equals(e1,e2)
step(e)
exp(e)
log(e)
inprod(v1,v2)
inverse(v)
max(e1,e2)
min(e1,e2)
ranked(v,s)
mean(v)
sum(v)
sd(v)
phi(e)
pow(e1,e2)
sqrt(e)
```

Note: there is no function for computing a product. (Except, in OpenBUGS there is). But the summation can be used for doing this, by taking logarithms: $ab = \exp(\log(ab)) = \exp(\log(a) + \log(b))$.

Remember: these logical functions calculate a deterministic value that must be assigned to some variable, "$\leftarrow$", and those variables must not have an assigned value as data, nor initial value. That would lead to 'multiple definition' errors.

Sometimes we need to have IF-THEN -structures, but these are not part of WinBUGS syntax in the same way as in procedural languages. Remember, WinBUGS is a declarative language. Therefore, if we need something like this:

$$\begin{array}{ll} \text{if } y = 1 & \text{then } x \sim N(\mu_1, 1) \\ & \text{else } x \sim N(\mu_2, 1), \end{array}$$

it has to be coded as:

```
x[1] ~ dnorm(mu[1],1)
x[2] ~ dnorm(mu[2],1)
z <- equals(y,1)*x[1] + (1-equals(y,1))*x[2]
```

This shows how to define mixture distributions by setting y as a Bernoulli-variable, although we could not use the above when z is given as observed data value. (Data should always be assigned to a stochastic node, defined by $\sim$). The Bernoulli probability would correspond to weight of the first mixture component, $\mathrm{N}(\mu_1, 1)$. Also `step`-function could be used. Moreover, multiple logical choices could be implemented by using `categorical`-distribution:

```
a ~ dcat(p[])
z ~ dnorm(mu[a],1)
```

In this case, data could well be assigned to z which is now a stochastic node.

Note: sometimes you may need to compute an expression that is undefined for some values, e.g. $1/X$. Now, if X has e.g. Poisson distribution model $X \sim \text{Poisson}(\theta)$, and θ is given as constant or random, it can happen that for some iterations $X = 0$. Computing $1/X$ would lead to runtime error. How to avoid this? If we had a procedural language, we could use IF-THEN structure by first calculating the value of X and only then choose what to calculate (or not) after knowing what X value we had. But in declarative language we need to express a definition using only logically structured expressions. For example:

```
Y <- equals(X,0)*(-9) + (1-equals(X,0))*(1/X))
```

It would seem that this solves the problem by setting an arbitrary value of -9 whenever $X = 0$, and only calculate $Y = 1/X$ when $X \neq 0$. But WinBUGS produces an error, because it calculates also the case $1/0$ even though it would be multiplied by zero (which still would be undefined $0/0$). This does not work! The only hope is to replace X by $X + \epsilon$ with a very small value for ϵ to ensure that WinBUGS can compute $1/(X + \epsilon)$ for all values of X. But this introduces small error when $X > 0$ (which may be insignificant). Alternatively, we really need to think over the modeling, and redefine a new model which does not involve even the possibility of $1/0$.

8.7.1 Example: hypothesis of two models

If the alternative hypotheses can be described as alternative models: $M_1 : N(0,1)$ and $M_2 : N(3,1)$ (with no other choices, so $P(M_1) = 1 - P(M_2)$) and observed data consists of $z = 1$, then the posterior probability of each hypothesis can be modeled as:

```
model{
a ~ dcat(p[])
p[1] <- 0.5; p[2] <- 0.5
z ~  dnorm(mu[a],1)
mu[1] <- 0; mu[2] <- 3
z <- 1
P1 <- equals(a,1)
}
```

By computing the average of `P1` over the simulation, we can approximate the posterior probability:

$$P(\text{M1} \mid z) = \frac{P(z \mid \text{M1})P(\text{M1})}{P(z \mid \text{M1})P(\text{M1}) + P(z \mid \text{M2})P(\text{M2})}.$$

This can also be interpreted as a classification problem. The observation z belongs to either 'class', so that the simulated average of `P1` is an approximation of the posterior probability of belonging to class 1.

8.8 Irregular data structure

Original data are often not in the form of a regular $n \times m$ matrix, but in the form of a ragged array. For example, different individuals can contribute a different number of measurements. Such data could be augmented by symbols of missing data "NA":

```
list(A=structure(
       .Data=c(7,NA,NA,
               9,6,3,
               2,NA,5),
       .Dim=c(3,3)))
```

Alternatively, we could code the data as a single vector, and use auxiliary indexing:

```
 list(A=c(7,9,6,3,2,5),
      person=c(1,2,2,2,3,3))
```

In the model, we can then use nested indexing to pick the right data value for the right model expression:

```
A[i] ~ dnorm(mu[person[i]],tau)
```

If the augmentation by NAs was used, WB will interpret these as missing data, and it will automatically sample these values as any other unknown quantities in the model. In other words: we will get a posterior distribution for the missing values. (Actually, this will be either prior or posterior predictive distribution). It is also possible to use offset-variables, but that technique is more prone to coding errors than the other two approaches.

8.9 Distributions

A list of all available distributions (and also logical functions) is found from the Help-menu. It may be noted that the list is not exhaustive catalogue of all distributions. Usually it is sufficient also when some other distributions are needed, because some distributions are related to each other so that random variables of one distribution can be transformed to random variables of another.

Sometimes, problems occur in special situations. For example, if a binomial distribution becomes defined with $N = 0$. This could happen when the data contain a list of sample sizes in a surveillance scheme and sometimes there have been no samples at all. The error can be avoided by removing such 'data' because there would be no loss of information. Another problem might occur if the binomial N is being estimated. In this case, initial value should be larger than zero, although the resulting posterior will cover also zero. There are some minor differences between different versions of WB which may result in different behaviour in extreme or unusual situations.

8.9.1 Permutation problem

Goal: to construct a random permutation distribution, so that we simulate n ones (and $N - n$ zeros) within a vector x of length N:

```
model{
p[1] <- n/N
x[1] ~ dbern(p[1])
for(i in 2:N){
p[i] <- ( n - sum(x[1:i-1]) )/(N-i+1)
x[i] ~ dbern(p[i]) }
}
list(N=10,n=4)
```

The 'trick' here for allocating the n items randomly among N places is to define a bernoulli process where the probabilities change at every step, depending on the outcomes of the previous steps. Note: vector x represents the outcome of sampling n items without replacement, whereas `x[]` $\sim$ `dmulti(p[],n)` would represent sampling n items with replacement, with $p_i = 1/N$. In both cases, $\sum x_i = n$. Note also that the single site updating scheme can become a problem when a jump to next value requires simultaneous change of two or more parameter components. For example, the above permutation model (and the multinomial model) requires that there is a fixed number of ones in vector x. Hence, it is not possible in MCMC to move to a valid new value of x by changing just one element of x. If any element is changed from 0 to 1, or vice versa, then some other element needs to be changed accordingly to keep the sum of ones unchanged at all MCMC iterations. This requires so called block updating, generally not available in WinBUGS. Therefore, the above models may only work when simulating x as a prior distribution ('forward' Monte Carlo), but not if posterior of x would be required.

8.9.2 Censored data

As a special case of distributions, censored observations may need to be modeled. This means that instead of an exact observation of X, we only know that $X > L$. Then, instead of the conditional density

$$\pi(X \mid \text{parameters})$$

we need the conditional probability:

$$P(X > L \mid \text{parameters})$$

These are frequently used in survival modeling where the observations are often censored from right $X > L$ or left $X < H$, or both $L < X < H$. In WB, this can be implemented using I-function. For example, with normal density:

```
x ~ dnorm(mu,tau)I(low,)
y ~ dnorm(mu,tau)I(,high)
z ~ dnorm(mu,tau)I(low,high)
```

Here, the limits `low` and `high` are not allowed to depend on the parameters to be estimated, `mu` and `tau`. The same can be written using 'one's trick'. For example:

```
x ~ dnorm(mu,tau)
one <- 1
one ~ dbern(pr)
pr <- step(x-L)
```

Here, 'one' is an instrumental bernoulli-variable which is defined as 'observed data' telling that $X > L$. Note: in OpenBUGS, 'I' has been replaced by symbol 'C'. ('C' for censoring).

Truncated distribution modeling is different from censored data. For example, a truncated normal distribution $\mathrm{N}(\mu, \tau)$ over the interval $[5, \infty]$ has a probability density that has the same functional form as the original density, apart from a different normalizing constant:

$$\frac{1}{\int_5^\infty \pi(x \mid \mu, \tau)\, \mathbf{d}x}$$

which depends on the unknown parameters μ and τ. If these parameters are to be estimated, the normalizing constant is not constant with respect to these parameters, and we need to include the correct truncated model somehow. (In OpenBUGS, they promise to develop this for more clarity in the future! There is already T-function for truncation!). In principle, customized own distributions can be defined in WB using 'zero's trick':

```
C <- 10000
for (i in 1:N){
 zeros[i] <- 0
 phi[i] <-   -log(L[i])+C
 zeros[i] ~ dpois(phi[i])
}
```

where `C` needs to sufficiently large so that `phi` would be always positive. The trick is based on observing that the probability of 'zero' in a Poisson model is $P(0) = \exp(-\lambda)$, i.e. $\lambda = -\log(P(0))$. When we replace the probability $P(0)$ as the self defined probability of data point `x[i]`, given here as `L[i]`, the required probability model is obtained in lieu of the 'probability of zero'. If the variable X is discrete, then $-\log(L)$ is automatically positive. Constant `C` is only needed if X is continuous and `L` represents the values of density function that can be larger than one. Then, $-\log(L)$ might not automatically be positive. Alternatively, own distributions could be implemented using 'one's trick':

```
C <- 10000
for (i in 1:N){
 ones[i] <- 1
 p[i] <- L[i]/C
 ones[i] ~ dbern(p[i])
}
```

where `C` has to be enough large so that p < 1. Again, if L is directly a probability (not a value of density function), then this works automatically without constant `C`.

In both tricks, the data are a set of numbers $\{x_1, \ldots, x_N\}$ so that the self defined distribution gives the probabilities (or probability densities) L_i for each data point, and this can be written out as an expression.

8.10 Graphics

Graphics in WB is very basic and there are no attempts to develop more sophisticated graphical tools within WB since there are many other more advanced software already. It is best to take the MCMC sample out from WB and then process the graphics elsewhere. The histogram plots in WB can be modestly edited, though. (use 'properties'). The best option for quicker graphical processing is to run BUGS from R, so that R-graphics is readily available.

8.11 Scripts

It may be convenient to run WB using scripts. In this way, we can avoid mouse clicking all the steps in every simulation. The script is a series of commands written in the specific file `script.odc`. This file could be for example:

```
display('log')
check('C:/kurssi/koemalli.odc')
data('C:/kurssi/koemallindata.odc')
compile(3)
gen.inits()
update(500)
set(parameter)
update(1000)
history(parameter)
gr(parameter)
coda(parameter,C:/kurssi/output)
quit()
```

The model code would be written in the file `koemalli.odc`. The script is run by running the file `backbugs.exe`. The specified series of tasks is then done, and the results are written in files, file names starting with 'output'. In OpenBUGS, the scripting language has been further developed.

8.12 More resources

First aid can be sought here:

- Example codes (Help → examples Vol I & II).
- Help → User manual.
- WB FAQ:
`http://www.mrc-bsu.cam.ac.uk/bugs/faqs/contents.shtml`
- WB mailing list archives:
`http://www.jiscmail.ac.uk/lists/bugs.html`

If nothing helps, you can try WB mailing list.

Moreover, there are some extensions as downloadables. For example, GeoBUGS is a special package for spatial modelling that is already included in WB1.4.1. (Menu: map).

WB development interface provides more downloadable extensions, such as Reversible-Jump MCMC.

`http://www.winbugs-development.org.uk/`

By using WBDev, it is possible to build your own 'hardwired' WB functions and distributions, that would otherwise run too slowly if coded as part of the model definition code.

Finally: Always document your BUGS code properly! There can never be too many comment lines within the code. Just like with any other programming, after 6 months you will not remember clearly what `xzpred2` is. If the model is going to be used by you or anyone else after a longer time, make sure the code is readable and understandable. After all, it is the mathematical model that should be uniquely and clearly defined. There can be many different BUGS implementations for the same model, and the implementation of computer code alone is not a sufficient documentation of the model.

8.13 Running WinBUGS/OpenBUGS from R

Check the following site for some instructions:

`http://www.stat.columbia.edu/ gelman/bugsR/runningbugs.html`

Also, see:

`http://cran.r-project.org/web/packages/R2WinBUGS/vignettes/R2WinBUGS.pdf`

Basically, you need to: install R, install package R2WinBUGS, possibly also BRugs (for OpenBUGS). R2WinBUGS should be easy to install `install.packages("R2WinBUGS")`. For OpenBUGS: `install.packages("arm")`, `install.packages("BRugs")` . Note also:

'BRugs provides an R interface on Windows machines to OpenBUGS . It works only under Windows and used to be available from CRAN, now it is located at the CRANextras repository'.

`http://www.stats.ox.ac.uk/pub/RWin/`

But there are more instructions and downloadables from the OpenBUGS Website.

Once you have installations done, write the model in a file called "m1.bug" in the working directory of R. For example:

```
model{
for(i in 1:n){
 x[i] ~ dbin(pr[i],10)
```

```
  pr[i] <- p[i]*ps
  p[i] ~ dunif(0,1)
 }
 ps ~ dunif(0,1)
 }
```

In R, load the installed package (`library("R2WinBUGS")` or `library("arm"), library("BRugs")`). Let's generate some data for the problem in R:

```
  n <- 200
  p0 <- runif(n,0,1)
  x <-  rbinom(n,10,p0*0.9)
  plot(p0,x)
```

This could represent 200 populations, each with different prevalence p_0, and a sample of size $n = 10$ taken from each, analyzed with test sensitivity of 0.9. Then, define data variables and parameters to be monitored, and generate some initial values in R:

```
 data <- list ("x","n")
 parameters <- c("p","ps")
 inits<-function(){list(p=runif(200,0,1),ps=0.5)}
```

The previous function will generate randomly initial values for `p` but set a default value of 0.5 for `ps`. The function is handy for generating a long list of values. Run WinBUGS and plot some results for comparison with the 'true values':

```
 res.sim <-bugs(data,inits,parameters,"m1.bug",n.chains=1,n.iter=2100,
                                                   n.burnin=100,n.thin=1);
 attach.bugs(res.sim)
 plot(density(ps))
 points(0.9,0,col="red",pch=15)
 par(mfrow=c(3,3))
 for(i in 1:9){
 plot(density(p[((i-1)*2000+1):(i*2000)]),main='',xlab=paste('p[',as.character(i),']'))
 points(p0[i],0,col="red",pch=15)  }
```

In this example, the marginal posterior densities are approximated from the sample by using `plot(density())` which is similar to the density plot in WinBUGS. But in R, much more can be done in processing and visualizing the results.

8.14 DIC in WinBUGS briefly

Model comparison is a broad topic in itself, and it would be more naturally connected with the (other broad) topics of 'sensitivity analysis' and 'model fit assessment' than WinBUGS. However, as DIC is also an available button in WinBUGS, it appears in many examples.

DIC = Deviance Information Criterion

It is used for model comparison: a lower DIC value indicates a better model. It is based on the concept of deviance: $D(y, \theta) = -2\log(\pi(y \mid \theta))$. If we have normal distribution model, with fixed variance, then (up to a constant factor) this is the same as the following statistics

$$T(y, \theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathrm{E}(y_i \mid \theta))^2$$

It may be seen that this is the familiar squared error. Hence, deviance is the generalization of squared error that can be applied more broadly. A small error means better fit. However, this should be compensated by the effective number of parameters used. With more parameters, a better fit is naturally expected. Posterior mean deviance is obtained approximately from the MCMC sample:

$$\bar{D}(y) = E(D(y, \theta) \mid y) \approx \frac{1}{k}\sum_{k=1}^{K} D(y, \theta^{(k)}),$$

where k is the index of MCMC-iterations. Likewise, we may compute the deviance by using an estimate of θ, namely its posterior mean: $\hat{\theta} = E(\theta \mid \text{data})$. Using this, we get:

$$\hat{D}(y) = D(y, \hat{\theta}).$$

$\bar{D}(y)$ describes better the errors of the model than $\hat{D}(y)$, because a fitted model (computed with $\hat{\theta}$) always fits the data better than a model computed with other parameter values θ. In contrast, $\bar{D}(y)$ computes the average error over all possible values of θ. The difference can be seen as the gain that can be achieved by fitting the model. Hence, the 'effective number' of parameters could be defined as the difference:

$$p_D = \bar{D}(y) - \hat{D}(y),$$

which represents the gain from estimating $\hat{\theta}$. Also, p_D could be interpreted as the number of unconstrained parameters in the model. How many parameters? Intuitively, every parameter counts if it will be estimated without prior constraints, or other constraining prior information. It counts null if it is completely constrained, or if all information about it comes from the prior. For example:

$$y \sim \mathrm{N}(\theta, 1) \quad , \quad \theta \sim \mathrm{U}(0, \infty).$$

In this model θ is constrained to be positive, but in other respects the prior is uninformative. How many parameters we have effectively depends on data y. If y is near zero, then practically half of the information comes from the prior and half from data, hence the model has about 0.5 effective parameters. But if y is very large, the prior constraint does not have any influence, and we effectively have one parameter. In hierarchical models, the number of effective parameters depends on the hyper prior. A very narrow prior will force all group specific parameters towards global mean, whereas a very loose prior lets them become independently estimated without much influence from other groups. In the latter case, the number of effective parameters is about the same as the number of groups.

Deviance Information Criterion (DIC) is available in WinBUGS/OpenBUGS, and applicable in some situations, but not in others (grayed out from the menu) and yet in some cases it may not be useful even when computed.

A better model is the one with smaller DIC-value, but difference less than 5 is said to be practically insignificant.

$$\mathrm{E}(D(y^{\mathrm{rep}}, \hat{\theta}(y))) \approx 2\bar{D}(y) - \hat{D}(y) = \mathrm{DIC}$$

Also:

$$\underbrace{\bar{D}(y)}_{\text{large is bad}} + \underbrace{p_D}_{\text{large is bad}} = \bar{D}(y) + (\bar{D}(y) - \hat{D}(y)) = 2\bar{D}(y) - \hat{D}(y) = \underbrace{\mathrm{DIC}}_{\text{small is good}}$$

'A model that would best predict a replicate dataset of the same structure as that currently observed'.

Example with multinomial model:

Use the following model with either small or large values for the prior parameter a, and check the effective number of parameters reported in BUGS ('pD').

```
model{
for(i in 1:4){a[i]<-250}
p[1:4] ~ ddirch(a[1:4])
x[1:4] ~ dmulti(p[1:4],N)
N <- 100
}
# data generated from p[i]=1/4,N=100:
list(x=c(26,29,19,26))
```

With very large a-values, most of the information is given in the prior, and posterior of p will be very focused at where the prior says. There's not much effect from data, hence, the number of parameters to estimate is effectively nearly zero, because we say we know the parameter values with high precision already in the prior. But if a is very small, then the prior is uninformative, and the result is dictated by data. In this case, all three parameters are freely influenced by the data, and 'pD' ≈ 3. (Note that the fourth parameter is always determined by the sum of the rest).

DIC does not always work and should be used with caution.

For example, the manuals says:
It is important to note that DIC assumes the posterior mean to be a good estimate of the stochastic parameters. If this is not so, say because of extreme skewness or even bimodality, then DIC may not be appropriate!

8.15 Exercises

1. Compute the approximate value of $\pi = 3.1415\ldots$ using the method previously described in WinBUGS. How many iterations are needed to get first 4 decimals correct?

2. Assume the model $X \sim \text{Bin}(N, 0.2)$, and that $X = 1$ was observed. Compute the posterior distribution of N in WinBUGS. Try different priors for N.

3. According to an expert, the sensitivity of a testing method is roughly in the range of $0.3 - 0.7$. Construct a beta prior distribution in WinBUGS that is approximately in this range.

4. Using the following data (generated from `dlnorm`(1,0.5)), fit `dlnorm`(μ, τ) and `dgamma`(α, β) models in WinBUGS. You may use these priors: $\mu \sim$N(0,0.001), $\tau \sim \Gamma$(0.001,0.001), $\alpha \sim$ exp(0.001) and $\beta \sim$exp(0.001). Another possible parametrization would be `dgamma`$(\mu\tau, \tau)$ with priors $\mu \sim$LN(0,0.001) and $\tau \sim \Gamma$(0.001,0.001) which enables to directly model the mean μ for the gamma distribution. Check the posterior of model parameters. Compute also predictive distribution for x and compare predictions. Compute DIC and compare models.

```
list(x=c(3.162, 0.5266, 1.245, 2.679, 1.781, 2.245,
       2.3, 9.147, 2.122, 12.53, 4.2, 18.95, 86.27,
       15.91, 6.062, 4.757, 4.911, 1.667, 2.749, 3.101))
```

5. The observed life times were

```
X=c(1.54, 0.70, 1.23, 0.82, 0.99, 1.33, 0.38, 0.99, 1.97, 1.10,
0.40)
```

and there were 4 censored observations at time $T = 2$. Assume $X_i \sim \text{Exp}(\theta)$ and prior $\theta \sim \text{Gamma}(2, 1)$. Write a WinBUGS model and compute the posterior of θ. Compute also the predictive distribution for a new observation, X^*.

6. Normal model with unknown mean μ, known variance σ^2. Compute in WinBUGS the posterior of μ assuming either small or large σ. Compare the results if the prior is either `mu` $\sim$ `dnorm(0,0.0001)` or `mu` $\sim$ `dflat()`.

```
 x=c(-0.7417224,-2.1873614,1.1508363,0.1306749,-1.1931158,0.2093445,-0.1040642)
```

7. Microbial samples are collected from individual animals and analyzed as pooled samples of 3 subsamples. (3 animal specific samples put together). Each pooled sample then results to either negative or positive test. A positive result is obtained if any of the 3 sub-samples were colonized. A negative result is obtained if none of the 3 sub-samples were colonized. Assume 50 pooled samples were analyzed and 1 was positive. Compute the posterior of pooled sample population prevalence, and the posterior of sub-sample (animal) population prevalence, using WinBUGS. Uninformative prior of animal prevalence can be used. What is the posterior probability that animal prevalence is >1%? Calculate posterior of both pooled and population prevalence for all possible outcomes: 0,1,2,...,50 positives in 50.

8. Proportion p of meals provided by a catering service are contaminated. Proportion q of consumed contaminated meals leads to illness. Assume uniform priors for (p, q). 300 meals were served in a conference and 5 people got sick. Compute in WinBUGS the posterior distribution of p assuming (1) $q = 0.5$ is known and assuming (2) q is unknown. When both parameters are unknown, plot their joint distribution. Extend the model with the unknown number of contaminated servings, Z, actually served. Compute the posterior predictive distribution for the number of illnesses in another conference with 100 people who were served the same food. Extend the model with experimental data in which 10 volunteers consumed contaminated meals and 2 of them got sick.

9. Consumers of broiler legs were given data loggers which measured the actual cooking time, t, and temperature, T, in the oven. The data show modestly negative correlation between t and T. Explain why this could be so. Compute the posterior distribution in WinBUGS assuming the following model for the logarithms. Compute also predictive distribution for a 'next' consumer. What is the probability that he/she will cook (A) less than 50 minutes, (B) over 50 min, but $T > 175$, (C) over 50 min, but $T \in [135, 175]$, (C) over 50 min, but $T < 135$? Is the model prediction realistic?

$$\begin{bmatrix} \log(t) \\ \log(T) \end{bmatrix} \sim \mathrm{N}\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \sigma_{11}^2 & \rho\, \sigma_{11}\sigma_{22} \\ \rho\, \sigma_{11}\sigma_{22} & \sigma_{22}^2 \end{bmatrix} \right)$$

$$\mu_i \sim \mathrm{N}(0, 0.001) \qquad \sigma_{ii}^2 \sim \mathrm{Gamma}(0.01, 0.01) \qquad \rho \sim \mathrm{U}(-1, 1)$$

Hint: use `inverse()` for computing the inverse covariance matrix in WinBUGS.

```
list(N=19, timetemp=structure(.Data=c(
    43,   179,
    44,   217,
    47,   206,
    49,   185,
    49,   166,
    53,   193,
    53,   180,
    56,   176,
    58,   167,
    59,   180,
    61,   132,
    62,   152,
    62,   136,
    72,   178,
    73,   168,
    78,   149,
    84,   169,
    99,   161,
   105,   148),.Dim=c(19,2)))
```

10. In 2001, 1962 Finnish voters were asked for their favorite political party. The result was

SDP	Kesk	Kok	Vihr	Vas	Other	Total
471	453	396	243	177	222	1962
24.0	23.1	20.2	12.4	9.0	11.2	100%

Using Dir(1,1,1,1,1,1)-prior and WinBUGS, compute the posterior density of the true population percentage for voters of each party. What is the probability that SDP was more popular than Keskusta?

11. Based on the results of the first 7 competitions of Ahonen and Janda, and uninformative prior, compute using WinBUGS the posterior probability $P(\mu_1 > \mu_2 \mid \text{data})$. Then, compute posterior predictive distribution for the result of the last competition for both jumpers. Knowing the total result of the first 3 competitions of the Four Hills Tournament, what is the probability that after the last competition, the difference of total points of the Tournament will be less than one point? Try predicting the whole Tournament, based on the four previous competitions.

12. Coal mine accidents in Britain, 1851-1962. Expand the model by using two or three change points. Compute the results in WinBUGS and use WinBUGS graphical tools to display them.

13. Use Multinomial model with Dirichlet prior to model the 2×2 table (Cornfield, 1962):

	Heart disease		
	yes	no	
Serum cholesterol < 260	51	992	1043
Serum cholesterol > 260	41	245	286
	92	1237	1329

If the cell probabilities are written $\begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix}$, the odds in the low chol group is $o_- = [p_{11}/(p_{11} + p_{12})]/[1 - p_{11}/(p_{11} + p_{12})] = p_{11}/p_{12}$. Likewise, the odds in the high chol group is $o_+ = p_{21}/p_{22}$. The odds ratio is o_+/o_-. The risk ratio is $RR = [p_{21}/(p_{21} + p_{22})]/[p_{11}/(p_{11} + p_{12})]$. Compute with BUGS the posterior distribution of odds ratio and risk ratio.

14. In the early BUGS, Gibbs samplers were thought to be coded directly by users. This is still possible in WinBUGS! Although, models should usually be defined as a structure of a DAG and ideally we should **not** get involved with sampling algorithm specifications. The whole point of WinBUGS is that users are free from such trouble and they could concentrate on model specification instead. Anyhow, consider bivariate normal density where (μ_1, μ_2) and the covariance matrix C are given,

$$C = \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}.$$

This could be simulated directly, or by using Gibbs sampler:

$$X_1 \mid X_2, \mu_1, \mu_2 \sim \text{N}(\mu_1 + \rho(X_2 - \mu_2), 1 - \rho^2)$$

$$X_2 \mid X_1, \mu_1, \mu_2 \sim \text{N}(\mu_2 + \rho(X_1 - \mu_1), 1 - \rho^2)$$

Implement both approaches in WinBUGS and compare. Notice that any pair of distributions $\pi(X_1 \mid X_2)$ and $\pi(X_2 \mid X_1)$ could thus be coded in WinBUGS, but they do not lead to a proper joint distribution $\pi(X_1, X_2)$ unless they are full conditional distributions that are correctly derived from the joint distribution. For example, what's wrong with `x ~ dnorm(x,1)`? Try it!