7 Monte Carlo method

A posterior probability distribution $\pi(\theta \mid X)$ captures all the information we have about the unknown parameter θ, after selecting the prior distribution $\pi(\theta)$ and the condition al distribution of data $\pi(X \mid \theta)$, and after observing what the data X was. *All results follow from the posterior distribution.* Usually, for the results we need to *integrate* over the posterior density. This can be simple to obtain from tabulated values of cumulative probability distribution (or from statistical functions on computer), when the posterior is among well known standard distributions - *as was the case with conjugate distributions.* Generally, this convenience is not available. Monte Carlo method is based on approximating the distribution by a sufficiently large sample from it. All we need to do is to find a way to draw random samples from the distribution. Our target distribution will naturally be the posterior distribution.

Assume we have drawn a sample $\theta_1, \ldots, \theta_n$ generated from the posterior distribution $\pi(\theta \mid X)$. Then,

- $E(\theta \mid X) \approx \frac{1}{n} \sum_{i=1}^{n} \theta_i$.

$\Rightarrow$ want to calculate posterior mean $E(\theta \mid X)$?

$\Rightarrow$ compute the sample mean $\frac{1}{n} \sum_{i=1}^{n} \theta_i$.

- $E(g(\theta) \mid X) \approx \frac{1}{n} \sum_{i=1}^{n} g(\theta_i)$.

$\Rightarrow$ want to calculate posterior mean of a transformation: $E(g(\theta) \mid X)$?

$\Rightarrow$ compute the sample mean $\frac{1}{n} \sum_{i=1}^{n} g(\theta_i)$.

- $P(\theta \in S \mid X) = E(1_{\{\theta \in S\}}(\theta)) \approx \frac{1}{n} \sum_{i=1}^{n} 1_{\{\theta \in S\}}(\theta)$

$\Rightarrow$ want to calculate posterior probability $P(\theta \in S \mid X)$?

$\Rightarrow$ compute the sample mean of the indicator variable $\frac{1}{n} \sum_{i=1}^{n} 1_{\{\theta \in S\}}(\theta)$.

Many standard distributions are available in statistical software as R, and we could also use those in WinBUGS/OpenBUGS. These could be used for simulating samples from given distributions, e.g. from posterior distribution that was found to be among standard distributions. This is always possible when using conjugate priors. Just plug in the appropriate parameter values for the standard distribution. With direct Monte Carlo method you can simulate any quantities of interest, and get approximate posterior means, medians, modes, and CIs.

For the conjugate models, you need the technical material about analytical solutions of posterior distributions given earlier for binomial, multinomial, poisson, gamma, and normal models.

So, for e.g. binomial model $\text{Bin}(N, r)$ with observed data x, and unknown r:

(1) if prior is $r \sim \text{Beta}(\alpha, \beta)$
(2) if data model is $x \sim \text{Binomial}(n, r)$
(3) then posterior is $r \sim \text{Beta}(x + \alpha, n - x + \beta)$
(4) use any software available to sample from this posterior distribution.
(5) monitor any quantity of interest, $g(r)$, and see the Monte Carlo sample histogram.

Price: for each problem, you need to solve the posterior probability density. This is unfortunately very restrictive, and the solutions for any more realistic problems become increasingly challenging.

7.1 MCMC

A much more general alternative to direct Monte Carlo sampling is Markov Chain Monte Carlo (MCMC). It is based on iterative approach where we start with some *initial values* θ_0, then sample next value conditional to that $f(\theta_1 \mid \theta_0)$, and continue sampling from $f(\theta_i \mid \theta_{i-1})$, $i = 1, 2, 3, \ldots$, where i denotes the ith sample of the parameter, the ith iteration step. The *transition distribution* f of the Markov chain is chosen so that, in the limit, the Markov chain converges to a stationary distribution, and this stationary distribution is the same as the distribution we want to draw samples from. A target distribution in Bayesian applications is naturally the posterior distribution. Hence, for each posterior distribution we are interested in, it is possible to construct a Markov chain sampler that will eventually draw random samples from it. There can be many different Markov chain samplers that will lead to the same target distribution but some are more efficient than others.

Note: the consequent samples are no longer independent and identically distributed as they are with direct Monte Carlo sampling where the next sampled value did not depend on the previously sampled value. Nevertheless, with sufficiently large number of iterations, we get approximately correct sample.

Note also: with all these Monte Carlo methods in Bayesian applications, the target distribution is the posterior distribution. It is not about simulating the biological random process. It is about simulating values according to the posterior distribution, i.e. describing our uncertainty distribution, given the observations we had from the biological process. An observation can be missing or predicted, in which case it becomes one of the unknown parameters among others, to be simulated by MCMC from the joint posterior distribution of all unknowns.

A special case of MCMC sampling is Gibbs sampling. This is sometimes called 'alternating' (vuorotteleva) sampling, because there we sample one of the unknown parameters at a time, from a distribution that is conditional to the current values of all other parameters and data. This is based on solving the 'full conditionals' from the joint distribution. For example with 2D-parameters θ_1, θ_2, we have the joint distribution as $\pi(\theta_1, \theta_2 \mid X)$. We can look at this in the 'proportional to' form, given by Bayes formula: $\pi(X \mid \theta_1, \theta_2)\pi(\theta_1, \theta_2)$. When you write down what these functions are in this product, look for an expression that is proportional to a familiar distribution for θ_1, given θ_2 and X. This should appear when you re-write the joint posterior probability of θ_1, θ_2 in the form $\pi(\theta_1 \mid \theta_2, X)\pi(\theta_2 \mid X)$. Then do the same to find a distribution for θ_2, given θ_1 and X. This should appear when re-writing the joint posterior as $\pi(\theta_2 \mid \theta_1, X)\pi(\theta_1 \mid X)$. If these two conditional distributions can be identified from the expression, you can use them to sample $\theta_1 \sim \pi(\theta_1 \mid \theta_2, X)$ and $\theta_2 \sim \pi(\theta_2 \mid \theta_1, X)$ sequentially: first θ_1, then θ_2, then θ_1, then θ_2...

For example: think again the binomial model, but let both X and p be unknown, and set $N = 20$

fixed. We try to compute the joint distribution of X and p, given $N = 20$, describing jointly the prior distribution of p and prior predictive for X. These are not independent because $\pi(X, p) \neq \pi(X)\pi(p)$, see also the figure shown in the section for binomial model. Here, we have N as fixed number in all calculations, so for simplicity it might be dropped from all notations; it is an underlying assumption here. The joint distribution of p and X is (according to product rule) written in two possible ways

$$\pi(p, X \mid N) = \underbrace{\pi(p \mid X, N)}_{\text{Beta}(X+1, N-X+1)} \underbrace{\pi(X \mid N)}_{*} = \underbrace{\pi(X \mid p, N)}_{\text{Bin}(N,p)} \underbrace{\pi(p \mid N)}_{**}$$

When you look at these two alternative expressions you can find that (1) if keeping X fixed in the first expression you have a distribution function for p proportional to $\text{Beta}(X + 1, N - X + 1)$. Likewise, (2) if keeping p fixed in the second expression, you have a distribution function for X, proportional to $\text{Binomial}(N, p)$. These are the full conditional distributions for p and X. MCMC sampler is given in R below. This joint distribution is the same as earlier when introducing binomial model and when simulating the joint distribution from the prior. There, the prior of p was uniform $\text{U}(0, 1)$ which is also the **marginal distribution of p**. This leads to the **marginal distribution of** X to be discrete uniform $\pi(X) = 1/(N + 1)$. From the joint distribution, these two marginal distributions can be written as ('**' and '*' above) $\pi(p) = \sum_X \pi(X, p)$ and $\pi(X) = \int_0^1 \pi(X, p)\mathbf{d}p$.

```
n<-20; p <- numeric(); x<- numeric()

p[1] <- 0.5  # initial values
x[1] <- 10
for(i in 2:1000){
p[i] <- rbeta(1,x[i-1]+1,n-x[i-1]+1)
x[i] <- rbinom(1,n,p[i])
}
plot(x,p)
```

From this we could actually compute also e.g. the posterior probability $\pi(p \mid X = 7, N = 20)$ by collecting all those iterations where we had $X = 7$. This is intuitive because we produced the joint distribution of X, p, and then we can condition to a specific value of X and see the distribution of p at that value of X. In principle, all posterior distributions might be simulated in this way: by sampling the joint distribution of the parameter and all possible data sets, and then collect those samples where the data coincides with our actually observed data X. This would be very inefficient and clumsy sampler, though. (But sometimes seen in some applications!). In practice, it is best to keep your data fixed to what it was, and only sample the quantities that are uncertain. (Posterior distribution is defined for those).

```
# R code for drawing the Gibbs sample:
 n<-20; p <- numeric(); x<- numeric()
p[1] <- 0.5; x[1] <- 5  # initial values
plot(x[1],p[1],xlim=c(0,20),ylim=c(0,1),
ylab="p",xlab="x",main="Gibbs sampling")
for(i in 2:250){
p[i] <- rbeta(1,x[i-1]+1,n-x[i-1]+1)
```

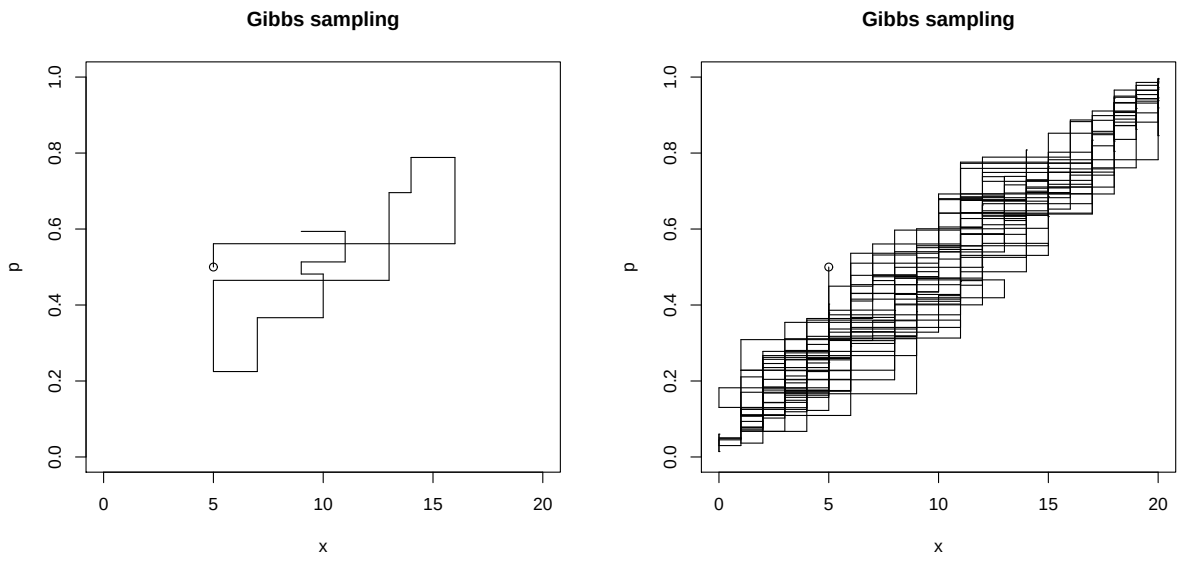


Figure 1: Sample path of Gibbs sampling with $\pi(X, p \mid N)$

```
points(c(x[i-1],x[i-1]),c(p[i-1],p[i]),'l')
x[i] <- rbinom(1,n,p[i])
points(c(x[i-1],x[i]),c(p[i],p[i]),'l')
}
```

Example: Gibbs sampling for a simple 2D normal density:

$$\begin{bmatrix} X \\ Y \end{bmatrix} \sim N\Big(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix} \Big)$$

Recall that the 2D normal density function (mean zero, unit variance) is

$$\pi(x, y) = \frac{1}{2\pi\sqrt{1-\rho^2}} \exp\Big(- \frac{1}{2(1-\rho^2)}(x^2 - 2\rho xy + y^2) \Big).$$

It can be written in the form $\pi(x \mid y)\pi(y)$ or $\pi(y \mid x)\pi(x)$ since the marginal, and conditional, densities can be solved from the joint density:

$$\pi(x) = \int_{-\infty}^{\infty} \pi(x, y)\mathbf{d}y = N(0, 1)$$

$$\pi(y) = \int_{-\infty}^{\infty} \pi(x, y)\mathbf{d}x = N(0, 1)$$

and

$$\pi(y \mid x) = \frac{\pi(x, y)}{\pi(x)}$$

$$= \frac{\frac{1}{2\pi\sqrt{1-\rho^2}} \exp\Big(- \frac{1}{2(1-\rho^2)}(x^2 - 2\rho xy + y^2) \Big)}{\frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}x^2)}$$

$$= \frac{1}{\sqrt{2\pi}\sqrt{1-\rho^2}} \exp(-\frac{1}{2(1-\rho^2)}(\rho x - y)^2) \quad = \quad \mathrm{N}(\rho x, 1-\rho^2)$$

and similarly: $\pi(x \mid y) = \mathrm{N}(\rho y, 1-\rho^2)$.

More general sampling algorithm is Metropolis-Hastings method, where within each iteration, the next sampled value is *proposed* from a proposal distribution. Then it is either rejected or accepted by a probability given by the Metropolis-Hastings ratio

$$R = \min\Big(\frac{\pi(\theta^* \mid \text{data})Q(\theta_{i-1} \mid \theta^*)}{\pi(\theta_{i-1} \mid \text{data})Q(\theta^* \mid \theta_{i-1})}, 1\Big),$$

where Q is the proposal distribution, x^* is the proposed new value and x_{i-1} is the current value from the previous iteration step. If θ^* is rejected, the previously sampled value θ_{i-1} is taken as the next value. The important innovation in this MH-ratio is that - again - the normalizing constant of the posterior is not needed. It cancels out from the ratio. Therefore, we only need to be able to calculate the posterior probabilities in the 'proportional to' form. Gibbs sampler is a special case of Metropolis-Hastings, where the acceptance probability becomes one, because the full conditional distributions are used.

Note: with all MCMC methods, the initial value can be chosen anywhere in the parameter space. The MCMC method is based on Markov Chain **that will only converge** to the correct target distribution. The Markov chain has a stationary distribution, and it is designed so that this stationary distribution is the same as the target distribution we want. But the Markov chain can be slow to converge. Generally, due to arbitrary initial values that may be far off the target distribution, it is common practice to run a *burn in* period. Only the sample collected after that will be used. The length of the burn in period needs to be judged separately in every case. There is no guarantee that a specific number of iterations is always enough.

WinBUGS = **B**ayesian inference **U**sing **G**ibbs **S**ampling

WinBUGS is a computer program designed for Monte Carlo simulation of posterior distributions, by using Markov Chain Monte Carlo methods. 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 is the version that is further developed, 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)`).

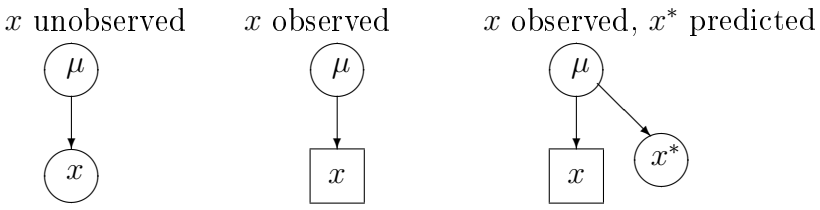
3. **Deterministic nodes** are logical functions of other nodes. Note that they are not allowed to be given data values. Data should always be given to a stochastic node as an observed value for that. Therefore, we cannot specify a structure where e.g. $X \sim \mathrm{N}(\mu, \tau)$ and $Y \sim \mathrm{N}(\mu, \tau)$ 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.

Some Directed Acyclic Graphs (DAG):



7.2 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!

7.3 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`)

7.4 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. That is: *all the conditional distributions and priors needed in the Bayes formula!*

Therefore, 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 set of all variables in the model can be a 'child node' that is conditionally dependent on its 'parent nodes'. And these 'parent nodes' can be 'child nodes' of other nodes, etc. By applying the product rule in the Bayes formula, a joint distribution of all variables V is broken into a product of conditional distributions:

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

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

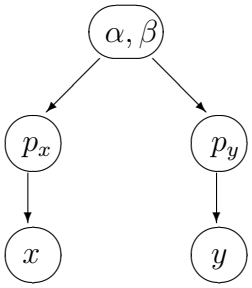
```
model{
x ~ dbin(px,n)
y ~ dbin(py,m)
px ~ dbeta(a,b)
py ~ dbeta(a,b)
a ~ dexp(1)
b ~ dexp(1)
}
```

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:



A DAG is really the visualized 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', but this is not a typical way of using it).

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.

7.5 Logical expressions

Long expressions can sometimes get too long for WinBUGS to compile. 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: $ab = \exp(\log(ab)) = \exp(\log(a) + \log(b))$.

Note: 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.

difficulty of IF-THEN programming

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

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

it has to be coded, for example, 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]
```

But 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 $\sim$. A logical node $\leftarrow$ within the code and a definition in the data listing for the same variable would cause multiple definition error). Therefore, if we want to compute posterior distribution of anything conditional to observed z, we should write something like

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

or using nested indexing, if y is binary variable:

```
z ~ dnorm(mu[ind],1)
ind <- y+1
```

Variable y could be indicator variable of the alternatives, for which we set Bernoulli(θ)-model with unknown parameter θ. This would correspond to probability of z being from the first model, $\mathrm{N}(\mu_1, 1)$. Also `step`-function could be used. This would make a mixture model for z:

$$\pi(z \mid \theta, \mu_1, \mu_2) = \theta \mathrm{N}(\mu_1, 1) + (1 - \theta)\mathrm{N}(\mu_2, 1)$$

where we need to choose prior distributions for all parameters θ, μ_1, μ_2. Moreover, multiple logical choices could be implemented by using `categorical`-distribution:

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

Another possible difficulty

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 $X \sim \mathrm{Poisson}(\theta)$, and θ is given as constant or random, it can happen that for some iterations $X = 0$. Maybe it is not sensible to define such variables that may lead to this problem anyway. But it could happen, for example if we have random N in a binomial model $X \sim \mathrm{Bin}(N, p)$ with $N \sim \mathrm{Poisson}(\lambda)$ and then try to define `Y <- X/N`, which could happen to be 0/0.

Computing $1/X$ with the possibility of $X = 0$ 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 was. 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.

7.6 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 Some BUGS models

Uncertainty with diagnostic testing

A diagnostic test has sensitivity $q_1 = P(\text{test} + \mid \text{true disease})$, and specificity $q_2 = P(\text{test} - \mid \text{truly no disease})$. Developers of the test have tested individuals that were first confirmed to be

healthy or diseased. These data can be used independently to compute posterior density of both parameters. Having this information, we can simultaneously compute posterior distribution of population prevalence from a sample whose testing results are observed. Priors for the three parameters are Uniform$(0,1)$. Run this model in OpenBUGS. Conditional distributions of data are:

$$X_1 \sim \text{Binom}(N_1, q_1) \quad , \; N_1 = 50, X_1 = 45$$

$$X_2 \sim \text{Binom}(N_2, q_2) \quad , \; N_2 = 30, X_2 = 28$$

$$Y \sim \text{Binom}(M, pq_1 + (1-p)(1-q_2)) \quad , \; M = 100, Y = 10$$

```
model{
x[1] ~ dbin(q[1],N[1])
x[2] ~ dbin(q[2],N[2])
y ~  dbin(pr,M);  pr <- p*q[1]+(1-p)*(1-q[2])
q[1] ~ dunif(0,1); q[2] ~ dunif(0,1); p ~ dunif(0,1)
}
list(x=c(45,28),N=c(50,30),M=100,y=10)
```

Comparison of two populations

Problem 1: to study whether the prevalence in one population is smaller than the prevalence in another population, based on a sample from both. The goal is to compute $P(\theta_1 < \theta_2 \mid \text{data})$. This is straightforward to simulate also in R. Assuming uniform prior density for both prevalence parameters, we draw samples from two beta-distributions, and calculate percentage of simulated values satisfying the requirement $\theta_1 < \theta_2$. In BUGS:

```
model{
x[1] ~ dbin(theta[1],N[1]); theta[1] ~ dunif(0,1)
x[2] ~ dbin(theta[2],N[2]); theta[2] ~ dunif(0,1)
# calculate average of T in the simulations:
T <- step(theta[2]-theta[1])
}
list(x=c(3,7),N=c(30,30))
```

Problem 2: similar as before, but now with normally distributed measurements. Here we compare population means

```
model{
# below only one measurement from both populations, but
# this could be expanded in a for-loop, to have x[i], y[i]
x ~ dnorm(theta[1],tau[1]); theta[1] ~ dnorm(0,0.001)
y ~ dnorm(theta[2],tau[2]); theta[2] ~ dnorm(0,0.001)
# calculate average of T in the simulations:
T <- step(theta[2]-theta[1])
}
list(x=3,y=4,tau=c(1,1))
```

Identifiability of parameters: an example

Consider two measurements $X_1 \sim \mathrm{N}(\mu_1, 1)$ and $X_2 \sim \mathrm{N}(\mu_2, 1)$. If we only observe the sum $Y = X_1 + X_2$ what can we infer about μ_1 and μ_2? To write a BUGS model, we have to invent a way to write Y as a stochastic node, so that it has a conditional distribution with some parameters. From probability theory of normal distributions we know that if $X_1 \sim \mathrm{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathrm{N}(\mu_2, \sigma_2^2)$ are conditionally independent, then $X_1 + X_2 \sim \mathrm{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$. This is our model. Assuming σ_i^2 are known, we aim to compute posterior distribution of μ_1 and μ_2. Clearly, these are not uniquely identified from data Y. The likelihood function is constant along a line $\mu_1 + \mu_2 = c$, so the individual parameters can take any values that are 'just as likely' combinations. A likelihood inference would conclude without a unique maximum likelihood estimate. Maximization algorithms would be trapped to never ending loops. BUT: bayesian inference could still get a solution, but this depends on priors. The priors must be informative in this case. The solution is sensitive to the choice of prior. **This is the danger with Bayesian methods: it is not always easy to notice identifiability problems because they might be covered up by the choice of prior**. If the problem remains in the posterior distribution, there should be also problems of convergence in MCMC sampling. Try the BUGS model:

```
model{
s ~ dnorm(mu,tau);  mu <- sum(m[1:2]);  tau <- 1/(sum(v[1:2]))
for(i in 1:2){ m[i] ~ dnorm(0,0.001);  # or more informative dnorm(0,1)
               v[i] <- 1/t[i]; t[i] <- 1
             }
s <- 3
}
```

Another identifiability problem: Eyes, WinBUGS Examples Vol2

Assume a set of observations X_i, some of them are from $\mathrm{N}(\theta_1, \sigma_1^2)$ and some are from $\mathrm{N}(\theta_2, \sigma_2^2)$. In this example, the mixture model likelihood becomes:

$$w\pi_1(X \mid \theta_1) + (1-w)\pi_2(X \mid \theta_2) = (1-w)\pi_1(X \mid \theta_2) + w\pi_2(X \mid \theta_1).$$

The likelihood function takes the same value if we switch the 'labels' (indexing) of θ and reverse the weights $(1-w)$ and w. This type of unidentifiability is called '*label switching problem*', or '*aliasing*'. The switching is eliminated by setting suitable constraints, in the implementation below it is ensured by defining θ_2 to be strictly greater than θ_1. Using the latent (unobserved) indicator variables, the model is:

$$\begin{array}{lcl} Z_i & \sim & \text{Bernoulli}(w) \\ X_i \mid Z_i & \sim & \pi(X_i \mid Z_i) \\ \theta_1 & \sim & \text{prior} \\ \theta_2 & \sim & \text{prior} \\ w & \sim & \text{prior} \end{array}$$

and in BUGS code of the Eyes-example:

```
model
    {
        for( i in 1 : N ) {
            y[i] ~ dnorm(mu[i], tau)
            mu[i] <- lambda[T[i]]
            T[i] ~ dcat(P[])
        }
        P[1:2] ~ ddirch(alpha[])
        theta ~ dnorm(0.0, 1.0E-6)I(0.0, )
        lambda[2] <- lambda[1] + theta
        lambda[1] ~ dnorm(0.0, 1.0E-6)
        tau ~ dgamma(0.001, 0.001) sigma <- 1 / sqrt(tau)
    }
```

Linear model: effect of standardization of covariates

Normal linear model is given by

$$y_i \sim \mathrm{N}(\beta X_i, \sigma^2)$$

where the expected value of observation y_i $(i = 1, \ldots, n)$ is a linear sum of the effects of explanatory variables $\beta X_i = \beta_0 + \beta_1 X_{i1} + \ldots + \beta_k X_{ik}$, so that X_i is a set of k explanatory variables for individual i. In this model, we have $k + 2$ unknown parameters for which we need to specify a prior. Often, uninformative priors are sought. One possibility is to use $\pi(\beta, \sigma^2) \propto 1/\sigma^2$ (improper prior). In the normal linear model with uninformative prior, the posterior density of regression parameters becomes (**conditionally to** σ) the following multivariate normal density:

$$\pi(\beta \mid y, X, \sigma) = N\Big(\underbrace{(X^T X)^{-1} X^T y}_{\text{mean vector}} , \underbrace{(X^T X)^{-1} \sigma^2}_{\text{cov. matrix}} \Big).$$

Typical BUGS-model of this linear model could be e.g.

```
model{
for(i in 1:n){
y[i] ~ dnorm(mu[i],tau)
mu[i] <- beta0 + beta1*x[i]
# mu[i] <- beta0 + beta1*(x[i]-mean(x[])) # standardized covariates
# or with more variables:
# mu[i] <- beta[1] + beta[2]*x[i,1] + beta[3]*x[i,2]
}
for(i in 1:k){
beta[i] ~  dnorm(0,0.001)
}
tau ~ dgamma(0.01,0.01)
# prediction with given value xnew:
ynew ~ dnorm(munew,tau); munew <- beta0 + beta1*xnew
```

```
# munew <- beta0 + beta1*(xnew-mean(x[])) # standardized covariates
}
list(y=c(41,52,18.7,55,40,29.2,51,17.6,46.6,57),
x=c(23.9,43.3,36.3,40.6,57,52.5,46.1,142,112.6,23.7),xnew=50)
```

In the above code, a prediction of `ynew` is included for a given covariate value `xnew` given in the data listing. This is the BUGS implementation of posterior predictive distribution $\pi(y_{\text{new}} \mid y_1, \ldots, y_n, x_1, \ldots, x_n, x_{\text{new}})$ which is **basically the same thing as before** in the examples where the predictive distribution could be analytically solved. In this case, the prediction is just based on the posterior distribution of all model parameters, and the assumed linear model with a given value of `xnew`.

Alternatively to the basic model $\beta_0 + \beta_1 x_i$, we can use standardized covariates: $\beta_0 + \beta_1(x_i - \bar{x})$ where $\bar{x} = \frac{1}{n}\sum_i x_i$. It can be useful to standardize the explanatory variables before modeling because this has an effect on the posterior covariance of parameters β. With the data below, compute posterior means in R according to the above multivariate normal density, and compute the covariance matrix (assume $\sigma = 1$) under original x variables, and under standardized variables $x_s = x - \bar{x}$:

```
y <- c(41,52,18.7,55,40,29.2,51,17.6,46.6,57)
x <- c(23.9,43.3,36.3,40.6,57,52.5,46.1,142,112.6,23.7)

X <- matrix(1,10,2)
for(i in 1:10){X[i,2]<-x[i]}
# posterior means for regression parameters beta:
betahat <- ((solve(t(X)%*%X))%*%t(X))%*%y
# covariance matrix of beta, assuming sigma=1:
C <- solve(t(X)%*%X)
```

This gives

$$\mathrm{C} = \begin{bmatrix} 0.346762974 & -4.269256e-03 \\ -0.004269256 & 7.386255e-05 \end{bmatrix}$$

```
# the same with standardized x:
xs <- x-mean(x)

Xs <- matrix(1,10,2)
for(i in 1:10){Xs[i,2]<-xs[i]}
# posterior means for regression parameters beta:
betahats <- ((solve(t(Xs)%*%Xs))%*%t(Xs))%*%y
# covariance matrix of beta, assuming sigma=1:
Cs <- solve(t(Xs)%*%Xs)
```

This gives

$$\mathrm{Cs} = \begin{bmatrix} 1.0000e-01 & -2.099300e-19 \\ -2.0993e-19 & 7.386255e-05 \end{bmatrix}$$

which is nearly a diagonal matrix, so that the correlations are now almost vanished. They should be exactly zero but the computer has limited accuracy. Analytically, we get

$$X_s^T X_s = \begin{bmatrix} n & \sum(X_{i2} - \bar{X}_{,2}) \\ \sum(X_{i2} - \bar{X}_{,2}) & \sum(X_{i,2} - \bar{X}_{,2})^2 \end{bmatrix} = \begin{bmatrix} n & 0 \\ 0 & \sum(X_{i,2} - \bar{X}_{,2})^2 \end{bmatrix} = \begin{bmatrix} 10 & 0 \\ 0 & 13538.66 \end{bmatrix}$$

Whereas computer gives something like

$$X_s^T X_s = \begin{bmatrix} 1.000000e+01 & 2.842171e-14 \\ 2.842171e-14 & 1.353866e+04 \end{bmatrix}$$

Anyhow, we can expect to obtain a posterior distribution of β with minimal correlations. (*This can be very useful in MCMC simulations of β such as Gibbs sampling*). The parameters of the standardized model and the original model are related, because:

$$\mu_i = \beta_0^* + \beta_1^*(x_i - \bar{x}) = \underbrace{\beta_0^* - \beta_1^*\bar{x}}_{\beta_0} + \beta_1^* x_i = \beta_0 + \beta_1^* x_i$$

Hence, with this standardization, β_j for $j \geq 1$ would remain the same, but β_0 of the original model would correspond to $\beta_0^* - \beta_1^*\bar{x}$ in the standardized model.

For curiosity, in BUGS, we could compute the same matrix, but since it is a function of data, it is a constant and cannot be monitored as an uncertain parameter. Its values can be checked using `Info` → `Node info` → `values`.

```
model{
for (i in 1:n) {
y[i] ~ dnorm(mu[i],tau.y)
xx[i] <- (x[i]-mean(x[]))
mu[i] <- a + b*xx[i] # standardized x
#mu[i] <- a + b*x[i]
X[i,1] <- 1
X[i,2] <- xx[i]
for(t in 1:2){XT[t,i] <- X[i,t] }
}
for(i in 1:2){for(j in 1:2){XTX[i,j] <- inprod(XT[i,],X[,j]) }}
XTXinv[1:2,1:2] <- inverse(XTX[,])
... ...
```