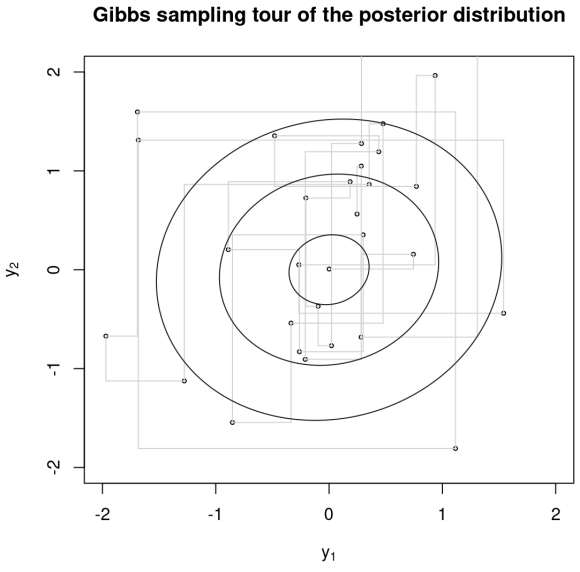
Practical 2. Markov Chain Monte Carlo - SOLUTIONS

Lecture 2 PDF version

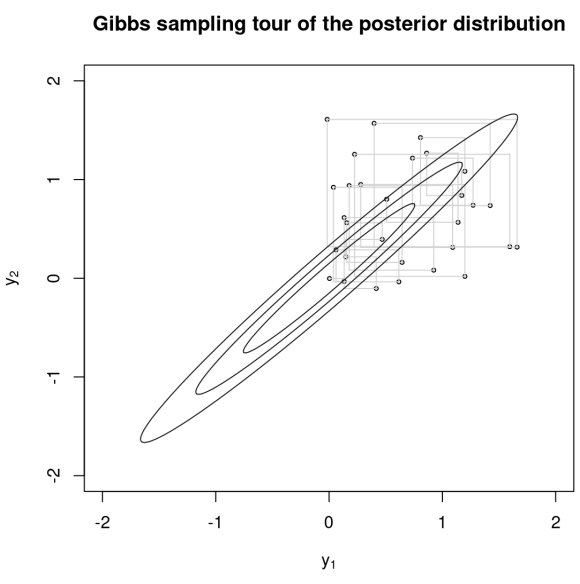
Understanding Gibbs sampling

Most of this exercise is actually fairly easy as it is performed directly in `Excel`. In fact, the model does not need MCMC to be analysed and we could obtain analytic estimates for the two variables (y_1, y_2). Some clarifications are perhaps useful, though.

The main point is perhaps to realise that correlation across the model parameters may be a crucial factor in terms of convergence and the speed at which this may be reached when running MCMC. This is due to the fact that larger correlation implies that the joint distribution of the parameters has a ``narrower'' shape. The figure below clarifies this point in the simplest two-dimensional, bi-variate Normal case — but the situation generalises to higher dimensions and different distributional shapes.



a. Low correlation $\rho = 0.08$



b. High correlation $\rho = 0.98$

The graphs show the first 30 simulations in a Gibbs sampling process for the same set up as in the `Excel` spreadsheet; the "true" underlying mean is $\boldsymbol{\theta} = (0, 0)$ and the "true" underlying variances are $\boldsymbol{\sigma}^2 = (1, 1)$. In the plot (panel a) we assume correlation $\rho = 0.08$ (i.e.~very low). This generates a joint distribution for (y_1,y_2)$ that is like a "ball", with a wide radius. This means that even a relatively low number of simulations (and more importantly long moves across the parametric space) is almost sufficient to cover the underlying true joint distribution.

In panel (b), however, because we are assuming a large correlation ($\rho = 0.98$), then the resulting distribution is very narrow and thus 30 simulations cannot cover the target portion of the space. In fact, in this case, the initial values make a difference — starting from values that are far awy from the bulk of the target distribution will generally mean that a longer running time is necessary.

MCMC in `BUGS`

As discussed in the lecture, the model is very similar to the one seen in Practical 1.

```
model {
   theta    ~ dbeta(a,b)         # prior distribution
   y        ~ dbin(theta,m)      # sampling distribution
   y.pred   ~ dbin(theta,n)      # predictive distribution
   P.crit  <- step(y.pred - ncrit + 0.5) # =1 if y.pred >= ncrit,
}
```

Again, we model our uncertainty on the underlying success rate for the drug using an informative, conjugate Beta prior distribution, which implies a mean of around 0.4 and a 95% interval spanning from 0.2 to 0.6. This time, however, we do have observed data on y — effectively now the objective of our analysis is estimate the *posterior* distribution of θ, $p(\theta \mid y)$ and to predict the outcome of the next round of the trial, when we assume to observed $n = 40$ patients alltogether.

Notice that in the model code, we are making the assumption that the new patients are effectively drawn from the same population of those that we have already observed. This is expressed by assuming that they have the same distributional form — both `y` and `y.pred` are modelled using a sampling distribution that is assumed to be Binomial$(\theta, \text{sample size})$ and effectively all that we assume to vary is the sample size ($m = 20$ for the observed y and $n = 40$ for the new patients).

The actual `.odc` file contains *all* the relevant bits that BUGS needs, in one place. In addition to the model code, it also contains the data, in the following format.

```
list(
a = 9.2, b = 13.8, # prior parameters
y = 15,            # number of successes
m = 20,            # number of trials
n = 40,            # future number of trials
ncrit = 25)        # critical value of future successes
```

This is a "list" (we shall see a lot more of this when we start working in R). As mentioned in the class, it is generally a good idea to keep the code general and then pass values that can be changed separately, rather than "hard-coding" them in the model code. In this simple case, however, you could "fix" the values of the parameters into the model code, as in the following.

```
model {
   theta ~ dbeta(a,b)                  # prior distribution
   y ~ dbin(theta,m)                   # sampling distribution
   y.pred ~ dbin(theta,n)              # predictive distribution
   P.crit <- step(y.pred - ncrit + 0.5) # =1 if y.pred >= ncrit,
   y <- 15                             #  0 otherwise
}
```

If you wanted to use this latter format, you would not need to pass BUGS any data, because all the relevant numbers would be included in the model code. Notice BUGS accepts a `` `double'' definition for the nodey` `` — first as a random variable associated with a Binomial distribution and then as a fixed number (15, indicating the observed number of successess in the current trial). In general, however, you need to be careful — for example it is not possible to define a node twice with the same nature (i.e. twice as observed data, or twice with two different probabilistic assignments).

Finally, the `odc` file contains the list for the initial values. In a BUGS file, all variables that are associated with a probabilistic statement (i.e. a "`~`" symbol after the variable name) **and** are *not* observed need to be initialised so that BUGS can run the Gibbs sampler. So, in this particular case, although `y` is defined as a random variable, because it is also observed, then there is no uncertainty about its value and so we should not initialise it. Conversely, although `y.pred` has exactly the same nature as `y`, because it is not observed, then we need to initialise it. In the `odc` file we actually let BUGS do this (effectively when clicking the `gen init` button in the `Specification tool`), but we do provide initial values for the other unobserved, random node, `theta`.

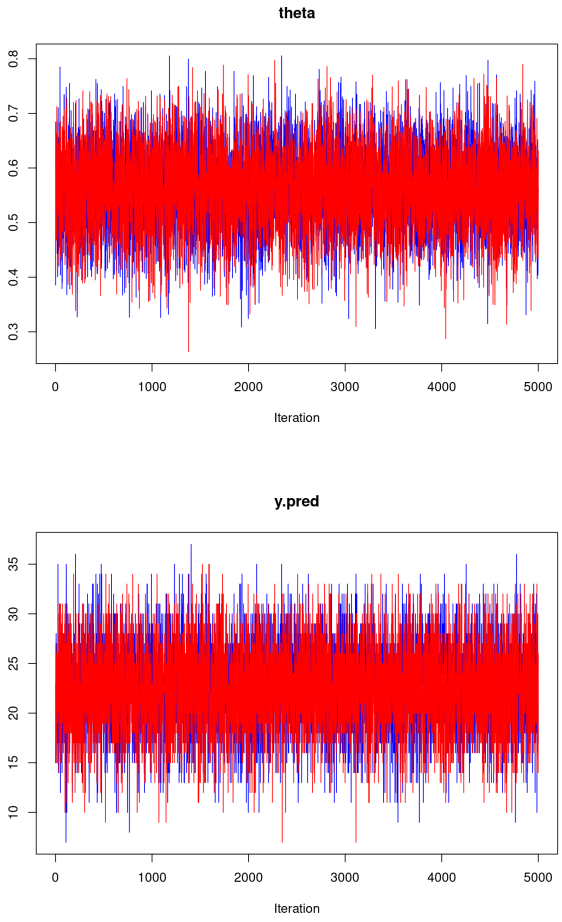
For the same reason mentioned above, BUGS does not let us monitor the node `y` — it has been already observed, so, as said above, there is nothing more to learn about it. It can be only used to learn about `theta`, `y.pred` and `P.crit`, which will all be associated with a posterior distribution, given the evidence provided by the fact that $y = 15$ out of $m = 20$ trials.

When we actually run the model, we can monitor the relevant nodes and produce the following summary statistics.

```
         mean     sd   2.5%    25%    50%    75%  97.5% Rhat n.eff
y.pred 22.578 4.2649 14.000 20.000 23.000 25.000 31.000    1 10000
theta   0.563 0.0749  0.414  0.512  0.563  0.614  0.706    1 10000
P.crit  0.330 0.4701  0.000  0.000  0.000  1.000  1.000    1 10000
```

These indicate that, given current evidence, we revise our uncertainty about the true success rate of the drug from a distribution centered on 0.4 and with most of the mass in [0.2; 0.6] to another centered around 0.563 and with most of the probability mass included in [0.414; 0.706].

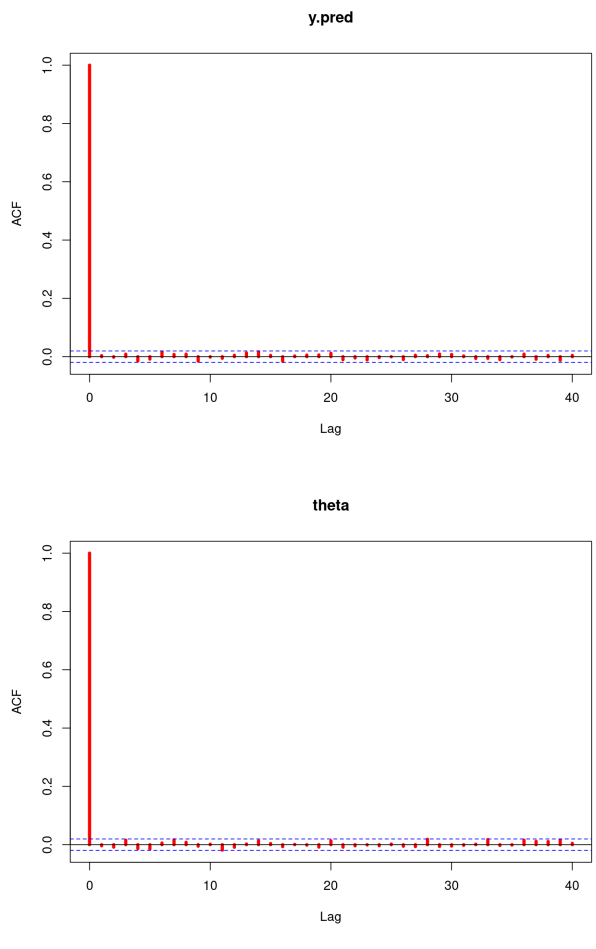
In fact, because we are using a Beta prior combined with a Binomial sampling distribution, then the model is *conjugated* and so we **know** that the form of the posterior distribution is another Beta (as seen in class), where the update from prior to posterior only involves the value of its parameters. This implies that we don't need to worry about convergence of the MCMC model. This is confirmed by the inspection of the traceplots.



In all cases, already at iteration 1 of the MCMC the two chains that we've run in parallel are on top of each other and evidently visiting the same portion of the parametric space.

As for `y.pred` and `P.crit` we can see that the expected number of successes in the next phase of the trial is around 23 and the probability of exceeding the critical threshold (25) is 32.97.

Notice finally that we can also look at the convergence statistics, $\hat{R}$ (`Rhat`, the Potential Scale Reduction, also known as "Gelman-Rubin Statistic") and the effective sample size n_{eff} (`n.eff`). The former has a value of 1 for all the nodes — again, given the fact that the model is conjugate, this is not surprising. The model does not present problems in terms of autocorrelation. The following graphs show the autocorrelation plot for the monitored nodes.



The way in which we need to interpret these is to look at the level of autocorrelation across different lags. These indicate how much correlation exists across consecutive, lagged simulations in the MCMC procedure. It is expected that consecutive iterations be relatively correlated (if you recall how the Gibbs sampling is constructed, you should realise that the $s-$th simulated value for the $p-$th parameter $\theta_p^{(s)}$ depends on the distribution at the $(s-1)-$th run for all the other parameters). But it is also expected that this level of correlation should fade away as you consider simulations that are further apart. So if the autocorrelation graph looks like the ones above, you can say that actually the simulations are close to a series of independent values because the autocorrelation is very low, in fact at low lags as well.

In our case, we that the model converges (because of conjugacy), so it is not surprising to see these graphs. Numerically, when the effective sample size is "close enough" the the nominal sample size (i.e. the number of iterations you use to draw your inference), then autocorrelation is not a problem. In this case, we have considered 2 chains, each with 5000 iterations (so a total of 10000). The value of `n.eff` is 10000 for `y.pred`, 10000 for `theta` and 10000 for `P.crit`. Clearly, the ratio of effective to nominal sample size is then 1, 1 and 1, which clearly indicate no issues.

NEXT
Covid vaccine: Bayesian modelling