Advanced Topics in Statistics

~ Introduction to MCMC ~

3

UNIVERSITY OF EXETER

RECAP.

Recall that Bayesian inference centres around the posterior distribution $p(\theta|\boldsymbol{x})$, and the interest usually lies in summaries of this posterior, which often take the form of integrals.

In the one parameter case we have already seen examples of problems where we were interested in the mean of the posterior distribution and tail areas (e.g. the probability that the parameter is larger than some critical value). Problems we have considered:

- What would an estimate of the flying bomb hit rate θ be in London during WWII? Theoretically this can be expressed as

$$E(\theta|\boldsymbol{x}) = \int_0^\infty \theta p(\theta|\boldsymbol{x}) \mathrm{d}\theta,$$

  where $p(\theta|\boldsymbol{x})$ is the posterior distribution of the hit rate.

- What is the probability that the absolute increase in major bleeds θ is less than 10 percent with low-dose PLT transfusions? This can be expressed as

$$P(\theta < 10) = \int_{-\infty}^{10} p(\theta|\boldsymbol{x}) \mathrm{d}\theta,$$

  where, again, $p(\theta|\boldsymbol{x})$ is the posterior distribution of the parameter.

MARGINAL POSTERIORS.

Even though so far we have only considered one-parameter models, Bayesian inference often involves complex models with a large number of parameters.

For multiple parameters Bayes Theorem gives us the joint posterior distribution

$$p(\boldsymbol{\theta}|x) \propto p(x|\boldsymbol{\theta}) \times p(\boldsymbol{\theta}),$$

where $\boldsymbol{\theta}$ is the vector of parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k)$.

The joint distribution describes the simultaneous behaviour of the parameters, but we usually also want to understand how the parameters behave individually.

It's the so-called marginal posterior that gives us the posterior distribution of a single parameter.

To obtain the marginal posterior we have to integrate out all the other parameters from the joint distribution. E.g. the marginal posterior of the parameter θ_i is given by

$$p(\theta_i|x) = \int \int \cdots \int p(\boldsymbol{\theta}|\boldsymbol{x}) \mathrm{d}\boldsymbol{\theta}_{(-i)},$$

where $\boldsymbol{\theta}_{(-i)}$ denotes the vector of θ's excluding θ_i.

Posterior summaries.

Once we have the marginals we can calculate properties the same way as in the one-parameter case. This again involves solving integrals.

For example, the mean and tail probability of the parameter θ_i are given by the following integrals:

$$\text{mean} = \int \theta_i p(\theta_i|x)\mathrm{d}\theta_i, \qquad \text{tail areas} = \int_T^\infty p(\theta_i|x)\mathrm{d}\theta_i.$$

Overall, we can see that many posterior integrals can be expressed in the form

$$\int g(\theta)p(\theta|x)\mathrm{d}\theta = E(g(\theta)|x),$$

for some integrable function g:

- For the mean $g(\theta) = \theta$.
- For tail probabilities $g(\theta) = \mathbf{1}_A$, where $\mathbf{1}_A$ is the indicator function of the appropriate tail area (in the previous example $A = (T, \infty)$).
- For posterior predictive densities we have $g(\theta) = p(\tilde{x}|\theta)$.

But other quantities of interest (e.g. marginals) are also expressed as appropriate integrals.

Monte Carlo integration.

We have seen that Bayesian inference heavily relies on the ability to compute integrals.

Integrals however can be analytically intractable even in relatively simple cases. Given the complex models, considering analytic solutions in Bayesian inferential problems is therefore often unrealistic.

Hence numerical integration is a vital part of Bayesian analysis.

Monte Carlo methods can be used to evaluate integrals numerically.

Suppose we can draw samples from the joint posterior distribution for $\boldsymbol{\theta}$, i.e.

$$\left(\theta_1^{(1)}, \ldots, \theta_k^{(1)}\right), \left(\theta_1^{(2)}, \ldots, \theta_k^{(2)}\right), \ldots, \left(\theta_1^{(N)}, \ldots, \theta_k^{(N)}\right) \sim p(\boldsymbol{\theta}|x).$$

Then

- Marginalisation is easy: $\theta_1^{(1)}, \ldots, \theta_1^{(N)}$ are samples from the marginal posterior $p(\theta_1|x)$.
- We can approximate integrals by the sample mean:

$$E(g(\theta_1)) = \int g(\theta_1)p(\theta_1|x)\mathrm{d}\theta_1 \approx \frac{1}{N}\sum_{i=1}^{N} g\left(\theta_1^{(i)}\right).$$

This latter is what we call Monte Carlo integration. Theorems exist which prove convergence to the integral in the limit as $N \to \infty$ even if the sample is dependent.

How do we sample from the posterior?

Monte Carlo integration relies on the ability to sample from the joint posterior $p(\boldsymbol{\theta}|x)$. Drawing independent samples from the posterior $p(\boldsymbol{\theta}|x)$ however may be difficult.

BUT dependent sampling from a *Markov chain* with $p(\boldsymbol{\theta}|x)$ as its *stationary (equilibrium) distribution* is easier.

- A sequence of random variables $\theta^{(0)}, \theta^{(1)}, \theta^{(2)}, \ldots$ forms a Markov chain if conditional on the value of $\theta^{(i)}$, the random variable $\theta^{(i+1)}$ is independent of $\theta^{(i-1)}, \theta^{(i-2)}, \ldots, \theta^{(0)}$.

  Intuitively this means that the future is independent of the past, given the present. (Or that we can make future predictions purely based on the present state).

  Note that, without any conditioning, the random variables of the Markov chain are not independent of each other.

- The stationary distribution of a Markov chain is a probability distribution (after a sufficiently long time) over the possible values of the Markov chain that remains unchanged as time progresses.

  In the discrete case, the stationary distribution gives the proportion of time, over the long run, that the chain spends in each state.

How do we sample from the posterior? - Gibbs sampling

Several standard 'recipes' available for designing Markov chains with the required stationary distribution. Gibbs sampling is an algorithm which generates a Markov chain by sampling from the full conditional distribution.

Gibbs sampling

Let our vector of unknowns $\boldsymbol{\theta}$ consist of k components $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_k)$.

1. Choose starting values $\theta_1^{(0)}, \theta_2^{(0)}, \ldots, \theta_k^{(0)}$.

2. Sample $\theta_1^{(1)}$ from $p(\theta_1 | \theta_2^{(0)}, \theta_3^{(0)}, \ldots, \theta_k^{(0)}, x)$.

   Sample $\theta_2^{(1)}$ from $p(\theta_2 | \theta_1^{(0)}, \theta_3^{(0)}, \ldots, \theta_k^{(0)}, x)$.

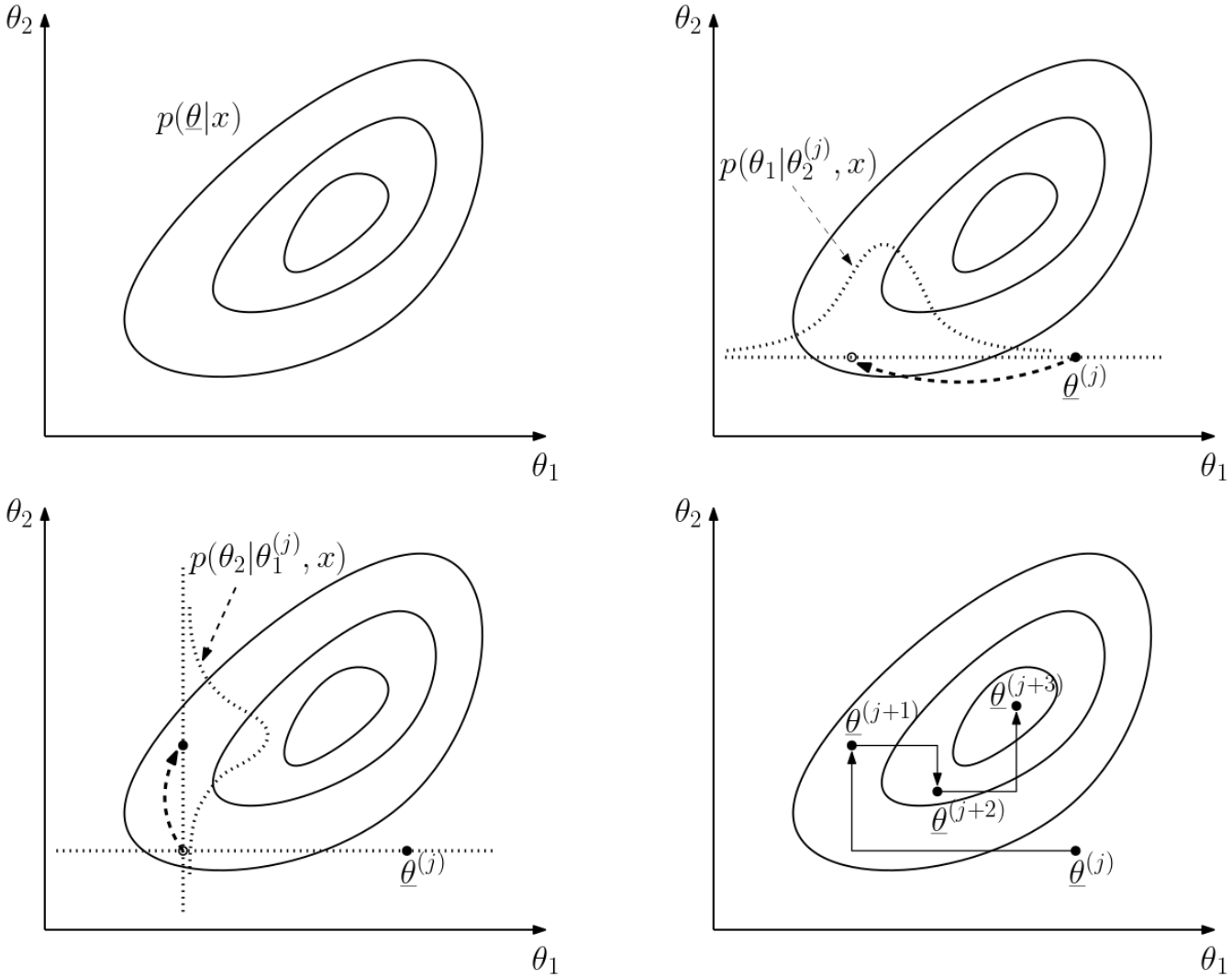
   $\vdots$

   Sample $\theta_k^{(1)}$ from $p(\theta_k | \theta_1^{(0)}, \theta_2^{(0)}, \ldots, \theta_{k-1}^{(0)}, x)$.

3. Repeat step 2 many thousands of times. Eventually we will obtain samples from $p(\boldsymbol{\theta}|x)$.

The conditional distributions are called 'full conditionals' as they condition on all other parameters.

GIBBS SAMPLING WITH $k = 2$.



$\boldsymbol{\theta}^{(n)}$ forms a Markov chain with (*eventually*) a stationary distribution $p(\boldsymbol{\theta}|x)$

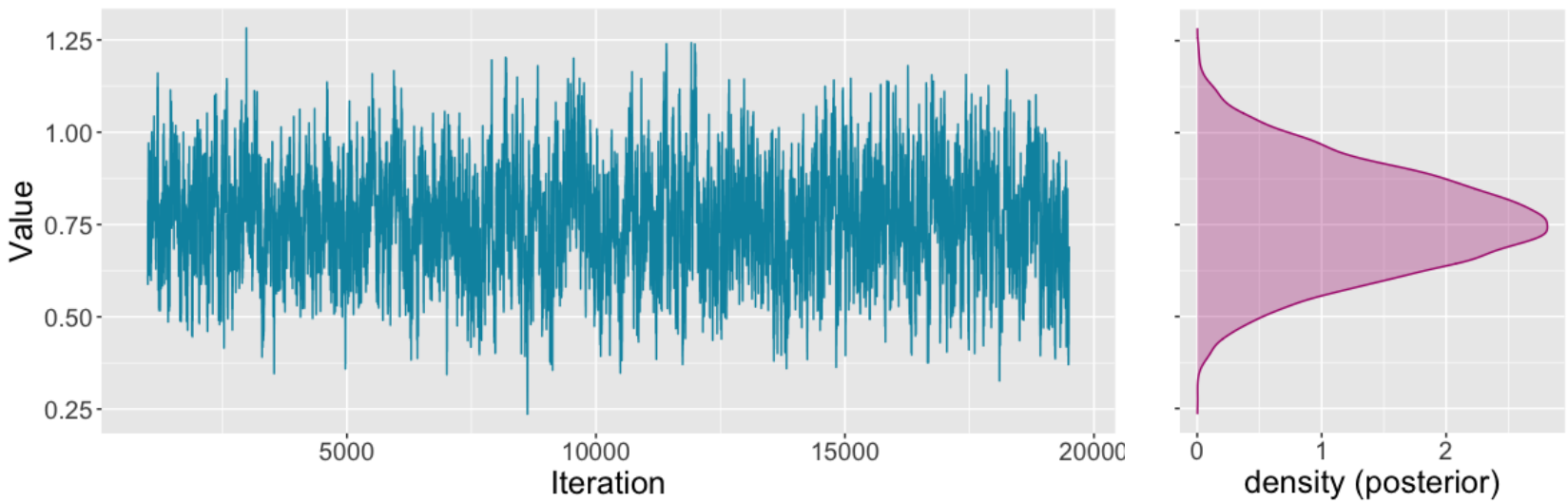
Convergence.

The main issue to consider when using Markov Chain Monte Carlo (MCMC) methods is convergence. (Note that convergence is to target distribution – the required posterior, not to a single value).

We only want to use those samples for the analysis that were drawn from the posterior.

Thus, we have to know *how quickly the distribution of* $\boldsymbol{\theta}^{(n)}$ *approaches* $p(\boldsymbol{\theta}|x)$.

Once convergence has been reached, samples should look like a random scatter about a stable mean value:



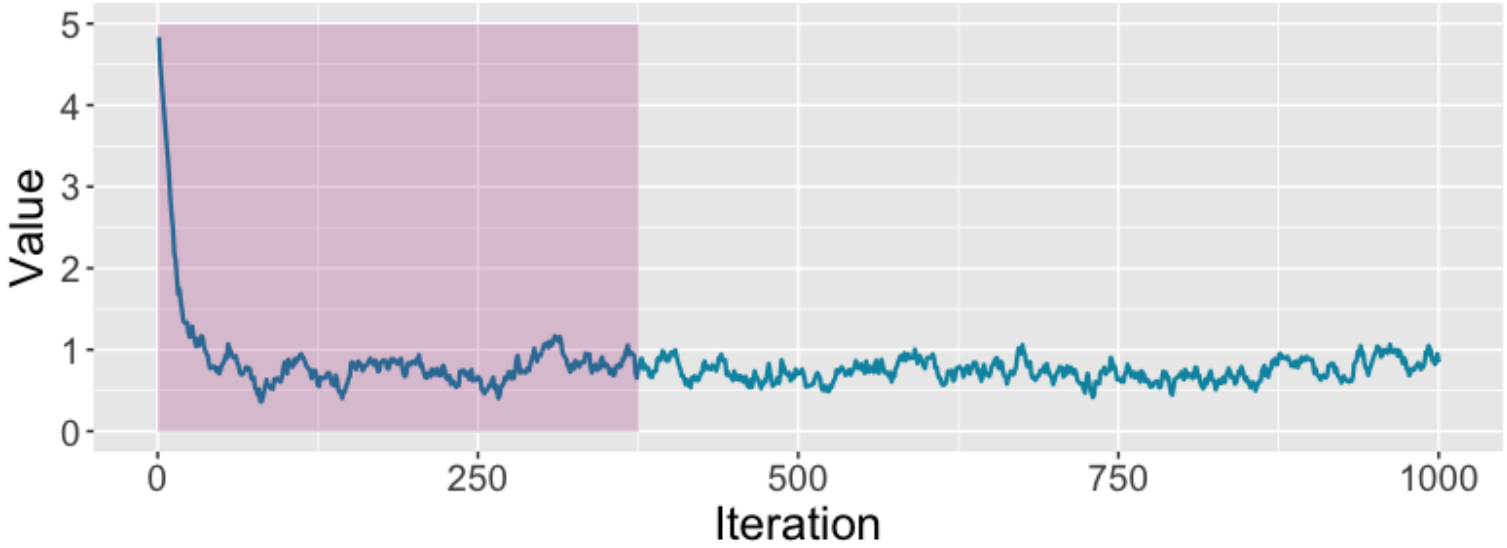
Convergence diagnostics.

Checking convergence is the user's responsibility!

To give time for the Markov chain to reach its stationary distribution, we can set a so-called burn-in period.

Samples produced by these burn-in iterations are discarded, and only those samples that were drawn after burn-in are saved for analysis.

But how do we know the number of burn-in iterations needed? That is, how do we know we have reached convergence?



Many 'convergence diagnostics' exist, but none is foolproof.

Gelman-Rubin diagnostic.

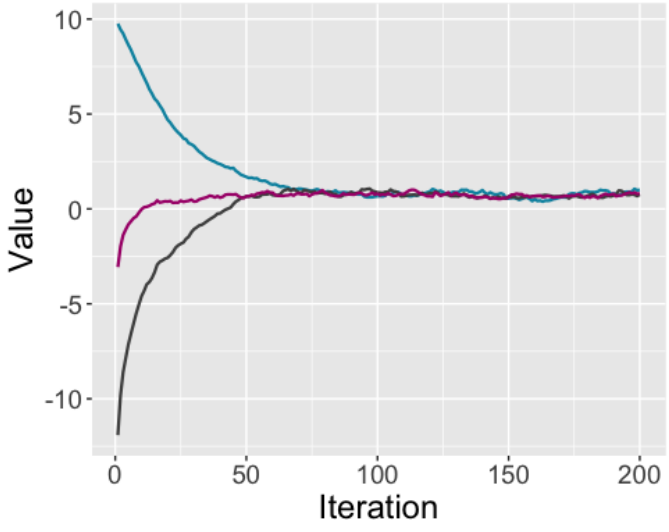
One of the most popular methods for checking the convergence of a Markov chain is the so-called **Gelman-Rubin diagnostic**.

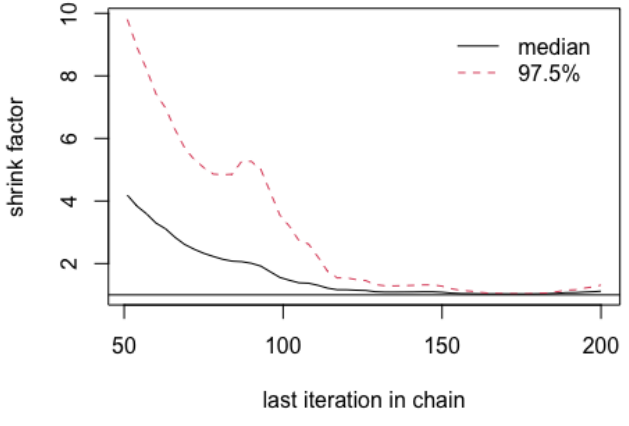
- To assess convergence we have to produce a number of chains.
- The chains should have widely differing starting points.
- Convergence is assessed by quantifying whether sequences are much further apart than expected based on their internal variability.

  I.e. Is inter-chain variability larger than within-chain variability?
- Diagnostic gives the so-called scale reduction factor for each parameter. A factor of 1 means that between chain variance and within chain variance are equal, larger values mean that there is still a notable difference between chains.

  As a rule of thumb, values above 1.1 indicate lack of convergence.

```
       Point est. Upper C.I.
theta        1.12        1.31
```





DOSE-RESPONSE MODEL EXAMPLE.

Consider the following response rates for different doses of a drug

dose x_i	No. of subjects n_i	No. of responses r_i
1.69	59	6
1.72	60	13
1.75	62	18
1.78	56	28
1.81	63	52
1.83	59	53
1.86	62	61
1.88	60	60

In order to explain the proportion of responses in terms of the dose, we fit a logistic curve with 'centred' covariate $(x_i - \bar{x})$:

$$r_i \sim \text{Binom}(p_i, n_i)$$

$$p_i = \frac{1}{1 + e^{-(\alpha+\beta(x_i-\bar{x}))}}$$

$$\alpha \sim N(0, 10000)$$

$$\beta \sim N(0, 10000)$$

```
jags.mod <- function(){
# likelihood
 for(i in 1:N){
   r[i] ~ dbin(p[i],n[i])
   p[i] <- 1/(1+exp(-(alpha + beta*(x[i]-mean(x)))))
 }
 # priors
 alpha ~ dnorm(0,1/10000)
 beta ~ dnorm(0,1/10000)
}
```

DOSE-RESPONSE MODEL EXAMPLE.

The data is stored in the vector `jags.data`, and we will monitor the nodes `alpha` and `beta`.

```
# data
jags.data <- list('x','r','n','N')

# parameters to monitor
jags.param <- c("alpha","beta")
```

To assess convergence we have to do multiple runs. Here we initialise 2 chains:

```
# initial values for 2 chains
inits1 <- list(alpha=-5, beta=10)
inits2 <- list(alpha=5, beta=-10)
jags.inits <- list(inits1,inits2)
```

and when fitting the model set `n.chain` to 2

```
# fit model
jags.mod.fit <- jags(data = jags.data, inits = jags.inits,
                     parameters.to.save = jags.param,
                     n.chain=2, n.burnin=0, n.iter = 10000,
                     model.file = jags.mod)
```

Note that the default value for the number of burn-in iterations is 500, which here we set to 0, meaning that all samples will be saved.

Output of dose-response model.

The command `print(jags.mod.fit)` gives the following summary

```
 2 chains, each with 10000 iterations (first 0 discarded)
 n.sims = 20000 iterations saved
      mu.vect sd.vect   2.5%    25%    50%    75%  97.5%  Rhat n.eff
alpha   0.767   0.139  0.500  0.673  0.764  0.860  1.044 1.001 20000
beta   34.720   2.910 29.249 32.726 34.612 36.634 40.650 1.001 15000

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
```

- Recall that the mean, standard deviation and the list of quantiles in the output are numerical summaries of the estimated posterior distributions.
- The `Rhat` value is an estimate of the scale reduction factor.
- Finally, `n.eff` is the effective sample size.

  Recall that the samples produced by the Gibbs sampling algorithm are usually not independent, but there's correlation between the observations (which is a function of the time lag between them). This concept is called autocorrelation.

  Intuitively, the effective sample size tells us the information content of the sampled values, or said another way, it gives what the size of the chain would be if the values were generated from independent sampling.

OUTPUT OF DOSE-RESPONSE MODEL - GELMAN.DIAG().

Another way to get estimates of the scale reduction factor is by using the `gelman.diag()` function of the `coda` package.

The function `gelman.diag()` not only gives us point estimates of the scale reduction factors, but also computes the associated confidence intervals, therefore it's a more reliable way of assessing convergence (than the Rhat values).

It needs an mcmc object as an input, so the fitted model has to be converted first. (Having an mcmc object also gives us additional options when plotting the output).

```
jagsfit.mcmc <- as.mcmc(jags.mod.fit)
```

Now we can get the Gelman-Rubin statistic values

```
gelman.diag(jagsfit.mcmc)
```

which gives

```
Potential scale reduction factors:

      Point est. Upper C.I.
alpha          1          1
beta           1          1
```

Having an upper C.I. close to one indicates convergence. As a rule of thumb we can use 1.1 (maybe 1.05) as a threshold.

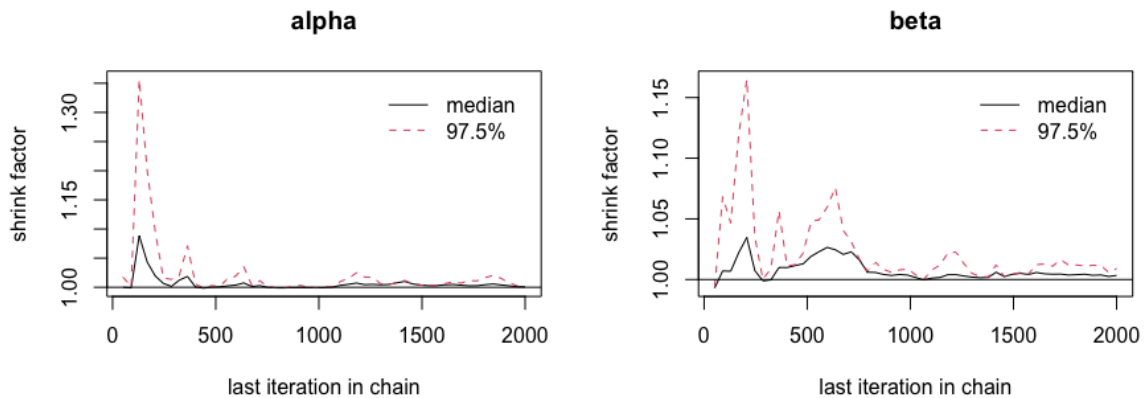
Output of dose-response model - gelman.plot().

The function `gelman.diag()` tells us whether convergence has been reached when we were fitting the model. But it doesn't tell us *at which point* convergence was reached.

Since in inferential problems we only want to use samples from the posterior, the observations drawn before the chains have converged should be discarded. But how do we know how many samples should we throw away?
Too little and we are risking including samples that don't represent the posterior, too many and we are wasting computing power.

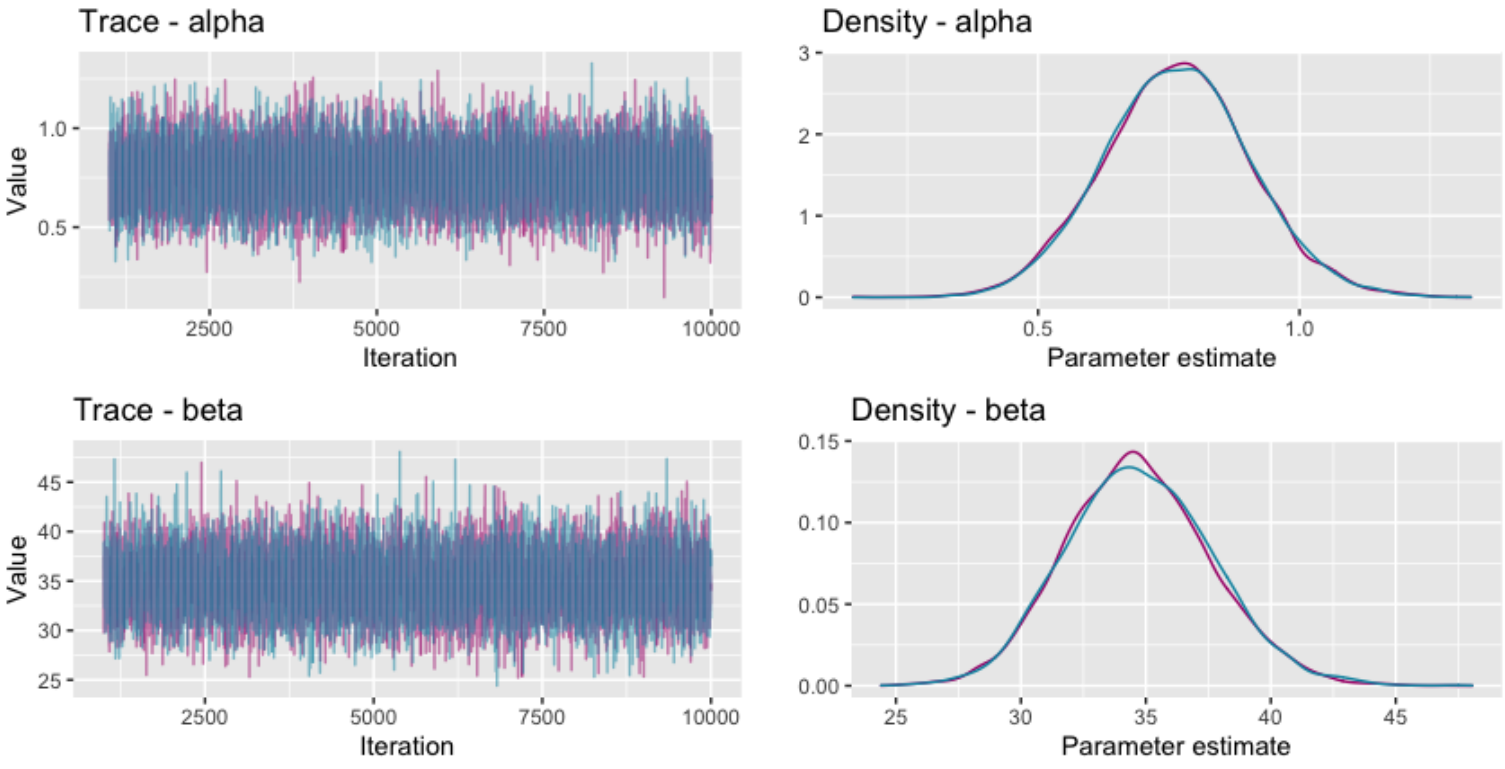
Plotting the evolution of the scale reduction factor as the number of iterations increases can help us choose a reasonable value for the number of burn-in iterations.

For the dose-response model (with 2000 iterations) the `gelman.plot` output implies that about 1000 iterations for burn-in should suffice.



Output of dose-response model - Traceplots.

Traceplots can be plotted using the `traceplot()` function or the `MCMCtrace()` function of the `MCMCvis` package (both of these take an mcmc object as an input), or manually after extracting the simulated samples.



The traceplots show good mixing, which is also an indication that the Markov chain has reached its stationary distribution.

Problems with convergence - Un-centred model.

Next we fit a logistic curve to the dose-response dataset with 'un-centred' covariate x:

$$
\begin{aligned}
r_i &\sim \text{Binom}(p_i, n_i) \\
p_i &= \frac{1}{1+\mathrm{e}^{-(\alpha+\beta x_i)}} \\
\alpha &\sim N(0, 10000) \\
\beta &\sim N(0, 10000)
\end{aligned}
$$

```
jags.mod <- function(){
# likelihood
 for(i in 1:N){
   r[i] ~ dbin(p[i],n[i])
   p[i] <- 1/(1+exp(-(alpha + beta*x[i])))
 }
 # priors
 alpha ~ dnorm(0,1/10000)
 beta ~ dnorm(0,1/10000)
}
```

Fitting this model gives the following output after 10000 iterations.

```
   2 chains, each with 10000 iterations (first 500 discarded)
 n.sims = 19000 iterations saved
      mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat n.eff
alpha -48.456  12.231 -62.530 -58.093 -50.538 -43.328 -15.092 1.148    18
beta   27.445   6.877   8.701  24.564  28.611  32.867  35.367 1.210    19

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
```

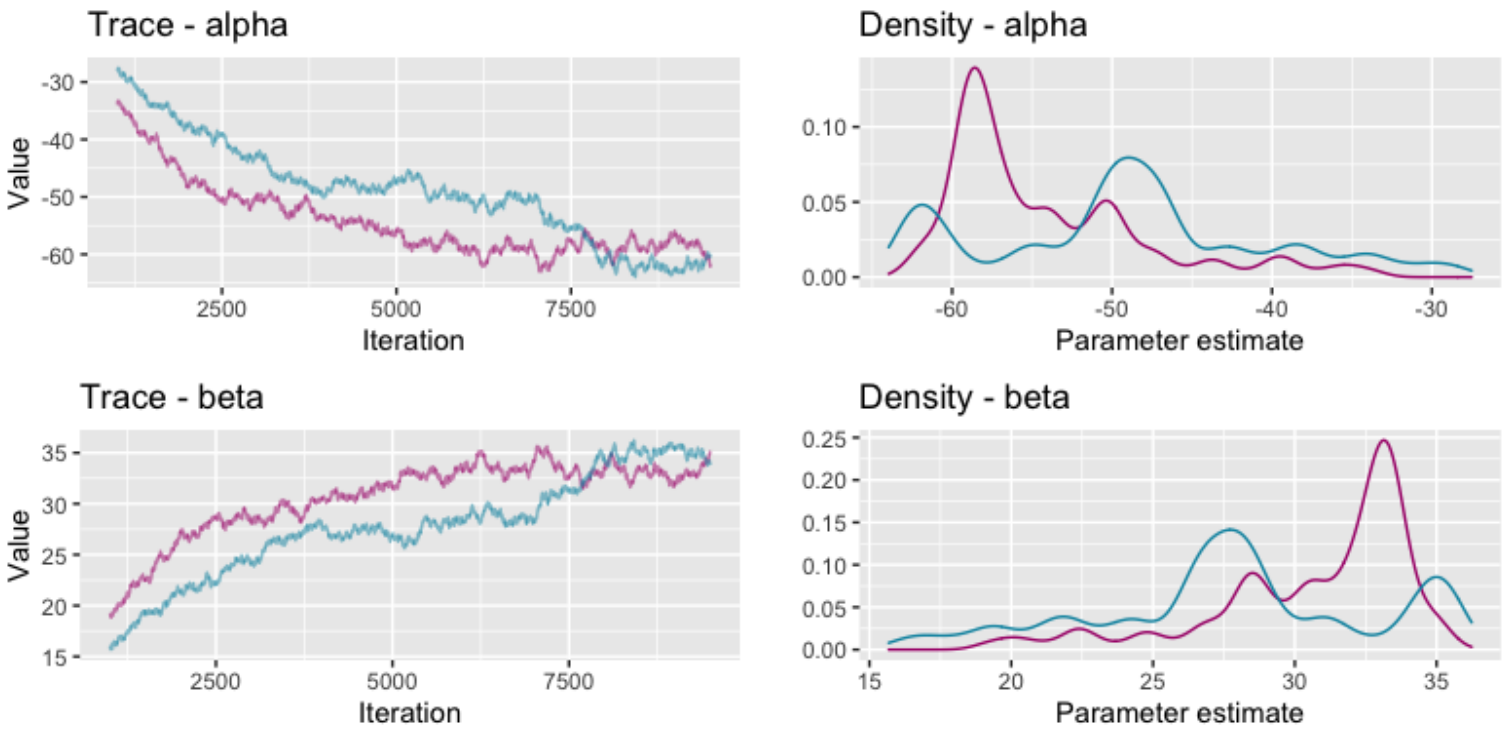
The Rhat values indicate lack of convergence, which we can be confirm by looking at traceplots and the `gelman.diag` output. (Note also the low effective sample size).

Output of un-centred model.

The output of `gelman.diag` confirms the lack of convergence, which is also apparent from the lack of mixing of the chains.

```
Potential scale reduction factors:

      Point est. Upper C.I.
alpha       1.54       4.62
beta        1.54       4.61
```



WHAT CAN WE DO WHEN THE MODEL HAS FAILED TO CONVERGE?

Lack of convergence can be a reflection of something fundamental with the model set up. E.g. trying to estimate a spatial correlation parameter when there is no strong spatial pattern.

If a model is non-identifiable, that will also result in chains not converging.

Therefore, as a first step, we should always sanity check the model.

If the model is correct and there's still no convergence we can try the following.

- Increase the number of iterations.
- Choose better initial values.
- Transform the variables.

  Centring, standardising covariates can reduce correlation between parameters, which helps with convergence.

We can also make use of some 'fast tracks' built into JAGS.

JAGS can not only recognise conjugacy, but also certain well-understood models – in both cases it can sample from a known distribution.

JAGS 'FAST TRACK' EXAMPLE.

JAGS can recognise generalised linear models when they are defined in a certain way.

As an example, consider the previous dose-response problem with uncentred covariates.

Recall that in JAGS we defined the link function as

```
p[i] <- 1/(1+exp(-(alpha + beta*x[i])))
```

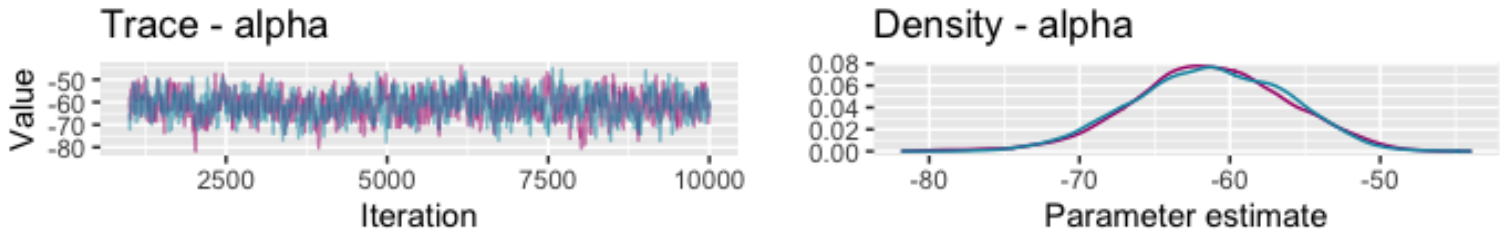
For logistic regression however, the link function can also be expressed using the logit function, where

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) = \alpha + \beta x.$$

In JAGS this translates to replacing the above logical node by

```
logit(p[i]) <- alpha + beta*x[i]
```

Using the logit function allows JAGS to recognise the glm, use a different sampling approach, which then in this case will result in convergence.



How many iterations after convergence?

After convergence, further iterations are needed to obtain samples for posterior inference. But how many samples should we draw?

- More iterations give more accurate posterior estimates, but we also have to keep in mind that drawing a large amount of samples is computationally expensive.
- The efficiency of the sample mean of θ as estimate of the theoretical posterior expectation $E(\theta)$ is usually assessed by calculating the Monte Carlo standard error (MC error).
- The MC error is the standard error of the posterior sample mean as an estimate of the theoretical expectation for a given parameter.
- MC error depends on the true variance of the posterior distribution, the posterior sample size (number of MCMC iterations), and the autocorrelation in the MCMC sample.
- As a rule of thumb we want the MC error to be smaller than 1-5% of the posterior standard deviation.

MC STANDARD ERROR IN JAGS.

In JAGS we need the `coda` package to extract the standard error.

- We first have to convert the fitted model to an mcmc object:

  ```
  jagsfit.mcmc <- as.mcmc(jags.mod.fit)
  ```

- Then the summary function of this mcmc object is what gives estimates of the standard error.

  ```
  summary(jagsfit.mcmc)
  ```

Note that the MC error is only meaningful for converged samples. Thus, before any inference, the model should be fitted using an appropriate burn-in period.

As an example, for the centred dose-response model (with `n.burnin=1000` and `n.iter=10000`) we get the following output.

```
1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean     SD Naive SE Time-series SE
alpha  0.7679 0.1396  0.00104       0.001444
beta  34.7203 2.9036  0.02164       0.030130
```

Since the Naive SE doesn't take into account the potential autocorrelation of the samples, this metric is not realistic.

Therefore, when assessing the MC standard error, we should use the Time-series SE values (and compare these with the SD values).

Inference using posterior samples from MCMC runs.

Once convergence is assessed, and we have enough samples to reach the required accuracy, we can move on to the inferential part of the Bayesian analysis.

A powerful feature of the Bayesian approach is that all inference is based on the joint posterior distribution.

Thus we can address a wide range of substantive questions by appropriate summaries of the posterior.

- We typically report either the mean or the median of the posterior samples for each parameter of interest as a point estimate.
- The 2.5% and 97.5% percentiles of the posterior samples for each parameter give a 95% posterior credible interval (interval within which the parameter lies with probability 0.95)

As an example considering the following output of the dose-response problem.

```
      mu.vect sd.vect   2.5%    25%    50%    75%  97.5%
beta   34.720   2.904 29.277 32.722 34.641 36.640 40.642
```

A point estimate of beta would be 34.72, with 95% credible interval (29.28, 40.64).

PROBABILITY STATEMENTS ABOUT PARAMETERS

Recall that classical inference cannot provide probability statements about parameters. E.g. the p-value is not $P(H_0 \text{ true}|x)$, but the probability of observing data as or more extreme than we obtained, given that H_0 is true.

In Bayesian inference however, it is simple to calculate e.g. $P(\theta > 1)$:

- Theoretically, this is the area under the posterior distribution curve to the right of 1.
- The theoretical value can be estimated by the proportion of values in the posterior sample of θ which are greater than 1.

In JAGS we can use the `ifelse` function to define a variable that takes the value 1 when the corresponding sample of θ is greater than 1, and takes the value 0 otherwise.

```
p.theta <- ifelse(theta>1, 1, 0)
```

The same function can be used in the discrete case.

For discrete parameters we might be interested in $P(\delta = \delta_0)$, for which we can use

```
p.delta <- ifelse(delta==delta0, 1, 0)
```

Then it's the posterior means of `p.theta` and `p.delta` that give the required probabilities.

COMPLEX FUNCTIONS OF PARAMETERS.

Classical inference about a function of the parameters $g(\theta)$ requires the construction of a specific estimator of $g(\theta)$. Obtaining an appropriate error can be difficult. (For this we would need to identify the distribution of $g(\theta)$).

However the problem is easy using MCMC. We just have to calculate the required function $g(\theta)$ as a logical node at each iteration, and summarise the posterior samples of $g(\theta)$.

As an example, in dose-response problem, suppose we want to estimate the dose, *ED95* that will provide 95% of maximum efficacy.

The required dose can be expressed using the link function. That is

$$\text{logit}(0.95) = \alpha + \beta(ED95 - \bar{x})$$
$$ED95 = (\text{logit}(0.95) - \alpha)/\beta + \bar{x}$$

Thus we just have to add the following to the model definition (and add *ED95* to the list of monitored nodes)

```
ED95 <- (logit(0.95) - alpha)/beta + mean(x)
```

which gives

```
      mu.vect sd.vect   2.5%    25%    50%    75%  97.5%  Rhat n.eff
ED95    1.853   0.008  1.839  1.848  1.853  1.858  1.870 1.001 18000
```

RANKING.

Ranking can be used in many scenarios. We might want to rank 'institutional' performance e.g. schools, hospitals; or certain treatments, while answer questions like 'which treatment is the best one'.

The rank of a point estimate however is a highly unreliable summary statistic. (E.g. sample size has a huge effect on point estimates).

Therefore, we would also like a measure of uncertainty about the rank.

Bayesian methods can easily provide posterior interval estimates for ranks.

Furthermore, the BUGS language contains 'built-in' options for ranking.

- `rank(x[])` transforms a vector `x` into a vector of ranks.
  E.g. if `y` $<-$ `rank(x[])`, then `y[i]` returns the rank of the ith element of `x`.
- `ifelse` can be used to find the element that is ranked the lowest.

  The mean of this gives a vector of probabilities, where entry i is the probability that the ith element is the 'best' (if counting adverse events).
- The function `sort` can be used to return the value of a certain ranked element.
  E.g. if `y <- sort(x[])`, then `y[i]` is the value of the ith ranked element.

RANKING - EXAMPLE.

The BUGS manual contains a random effects meta-analysis of 22 trials of beta-blockers. These were used to prevent mortality after myocardial infarction.

We will use the treatment arms of this data to rank the trials and find the one that is the most likely to have the lowest mortality rate.

Trial	n_i	r_i	Trial	n_i	r_i
1	28	3	12	263	45
2	144	7	13	291	9
3	69	5	14	858	57
4	1533	102	15	154	25
5	355	28	16	207	33
6	59	4	17	251	28
7	945	98	18	151	8
8	632	60	19	174	6
9	278	25	20	209	32
10	1916	138	21	391	27
11	873	64	22	680	22

- The vector n gives the number of patients treated with each drug.
- The vector r gives the number of patients who died in each trial regardless of the treatment.

BETA-BLOCKER EXAMPLE.

Notice how the number of patients involved varies a lot across the trials. This means that, when doing classical inference, the uncertainty associated to the point estimates of the mortality rates (r_i/n_i) are much smaller for some trials than for others.

The Bayesian approach takes the varied uncertainty associated to the trials into account when ranking the treatments.

In our model we will assume an independent Jeffreys' Beta$(0.5, 0.5)$ prior for each response rate (which is considered to be a vague prior in this situation), thus we have

$$\begin{aligned} r_i &\sim \text{Binom}(p_i, n_i) \quad \text{(likelihood)}, \\ p_i &\sim \text{Beta}(0.5, 0.5) \quad \text{(prior)}. \end{aligned}$$

Adding a node for the rank and another for the probability that the given trial has the lowest mortality rate results in the following model definition in JAGS.

```
jags.mod <- function(){
  for(i in 1:N){
    r[i] ~ dbin(p[i],n[i])
    p[i] ~ dbeta(0.5,0.5)
  }
  rnk <-  rank(p)
  prob.lowest  <- ifelse(rnk==1,1,0)
}
```

When fitting the model, we monitor the nodes `rnk` and `prob.lowest`.

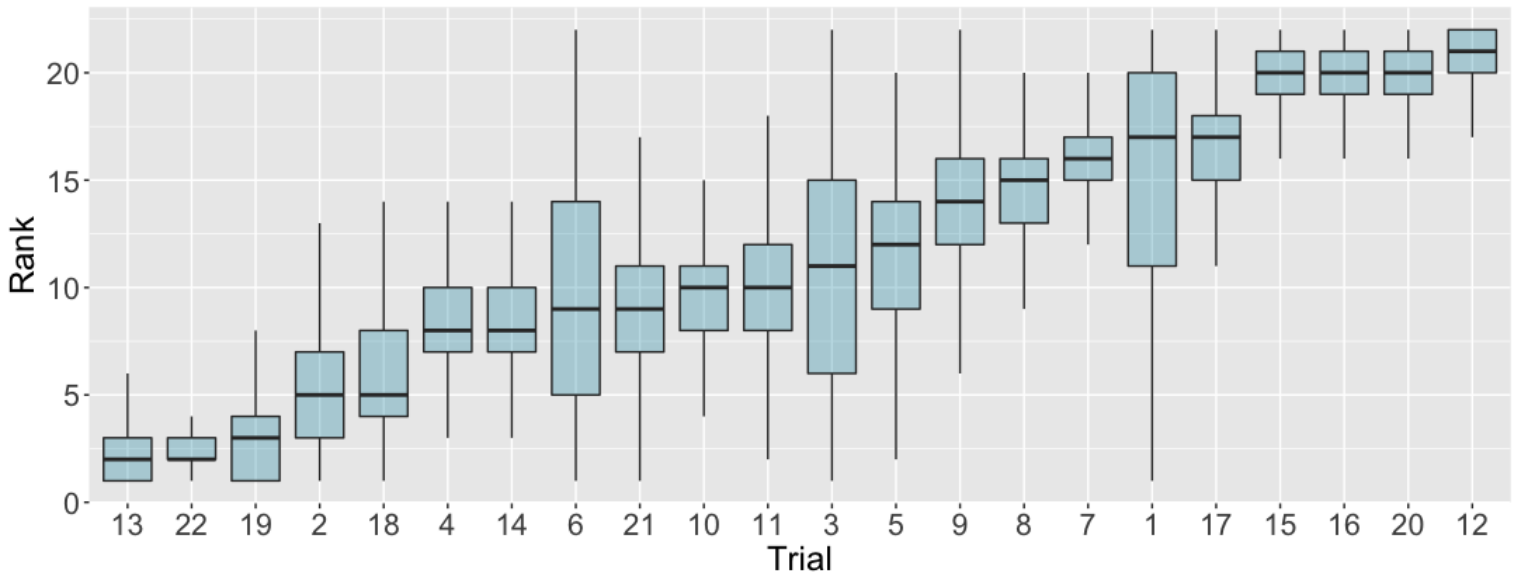
BETA-BLOCKER - RANK.

A boxplot is a good visualisation tool when considering ranks. It displays the median rank of each trial with the associated uncertainty.

After extracting the simulated samples from the fitted model, we can use `ggplot` to create a boxplot ordered by the median values of the rank.

```
r <- as.data.frame(jags.mod.fit$BUGSoutput$sims.list$rnk)

r %>% gather(key='trial',value='rank') %>%
  ggplot(aes(x=reorder(trial,rank,median),y=rank)) +
  geom_boxplot()
```



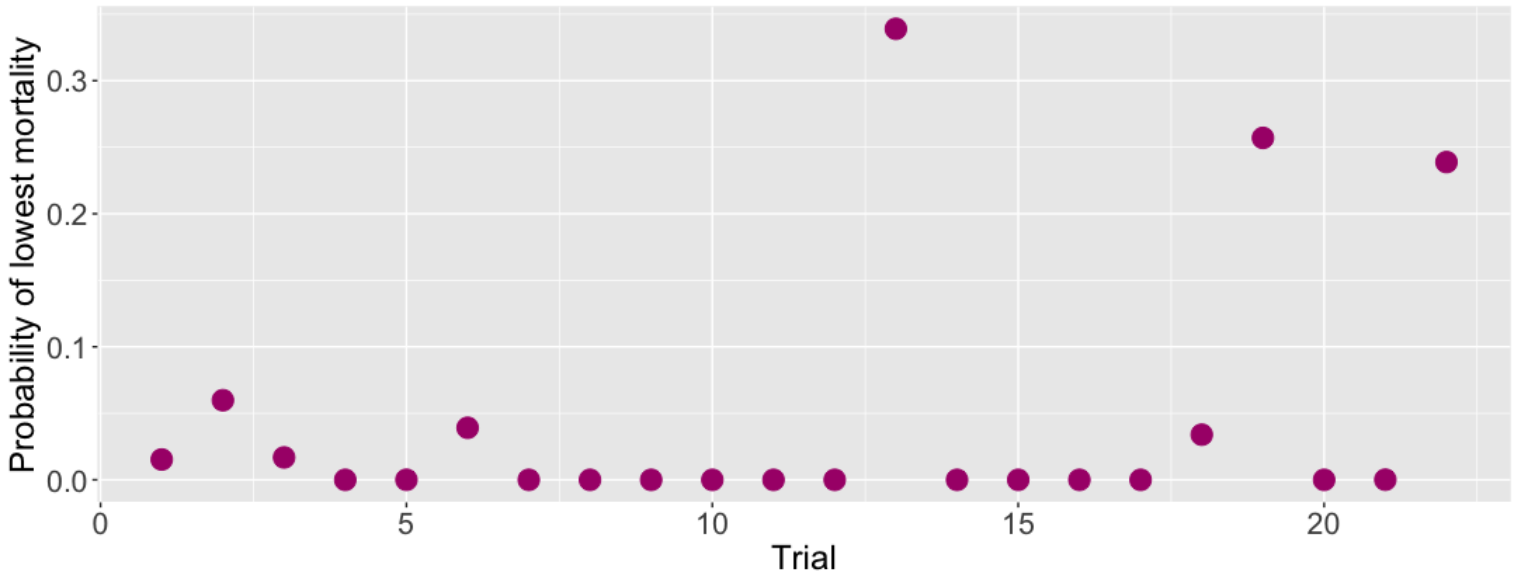
BETA-BLOCKER- PROBABILITY OF LOWEST MORTALITY.

A scatterplot can be used to display the probabilities of lowest mortality. Recall that

`prob.lowest[i]` $= P($trial i has the lowest mortality rate among the 22 trials$)$.

For this, we extract the mean of the posterior distribution of the `prob.lowest` node, and again, use `ggplot`.

```
prob.lowest <- jags.mod.fit$BUGSoutput$mean$prob.lowest
```



Thus trial #13 has the highest probability of being the trial with the lowest mortality rate (among the 22 considered trials).