MA40198: Applied Statistical Inference Assessed Coursework 2018

Candidates 20965, 20949 and 20969

13/12/2018

Part 1: Carcinogenesis Study On Rats

The data used in this section can be downloaded from the following location:

```r
rats <- read.table("http://people.bath.ac.uk/kai21/ASI/rats_data.txt")
rats$litter <- as.factor(rats$litter)

set.seed(743625723) # Set a random seed for reproducibility
```

The following packages are used to produce certain plots in this document.

```r
install.packages(c("ggplot2", "psych", "purrr"))
```

Section 1.1: Maximum likelihood estimation of distribution parameters

Our task here is to model the effect of a drug administered during a clinical trial. Specifically, we are interested in the time until tumour appearance in rats, where some of these rats had received a treatment of the drug. In essence, this is a survival analysis task, and in such cases the Weibull distribution is a common choice on which to base initial models.

However, before attempting to produce any models it is always sensible to examine the data for any obvious patterns. We do not have space in this report to include a detailed exploratory analysis, but we can consider simple density plots of the uncensored observations, separated by whether or not treatment was received (see Figure 1). Note that the distribution of the "treated" observations is possibly skewed slightly further to the left than the "untreated" distribution (i.e. tumour appearance tends to be sooner in treated observations); but given the small sample size, this could just as easily be explained by natural variance in the data as it could by any potential effect of the treatment.

We now assume that the times until tumour appearance, T_i, are distributed as

$$T_i \sim \text{Weibull}\left(\text{shape} = 1/\sigma,\ \text{scale} = \exp(\eta_i)\right), \qquad \eta_i = \beta 0 + \beta 1 x_i$$

where

$$x_i = \begin{cases} 1 & \text{rat } i \text{ received treatment} \\ 0 & \text{rat } i \text{ received placebo} \end{cases}$$

Initially, we will attempt to determine the values of the parameters of this distribution via maximum likelihood estimation. To do so, we construct expressions for the negative log-likelihood and its gradient, making use of R's `deriv()` function to calculate the derivatives of the Weibull probability density function and survival function (the latter will be used for censored observations, i.e. observations where the rat died before tumour appearance).

Note that we work with log-likelihoods throughout, and also that we have reparameterised σ with a log transform since we know this parameter must be positive.

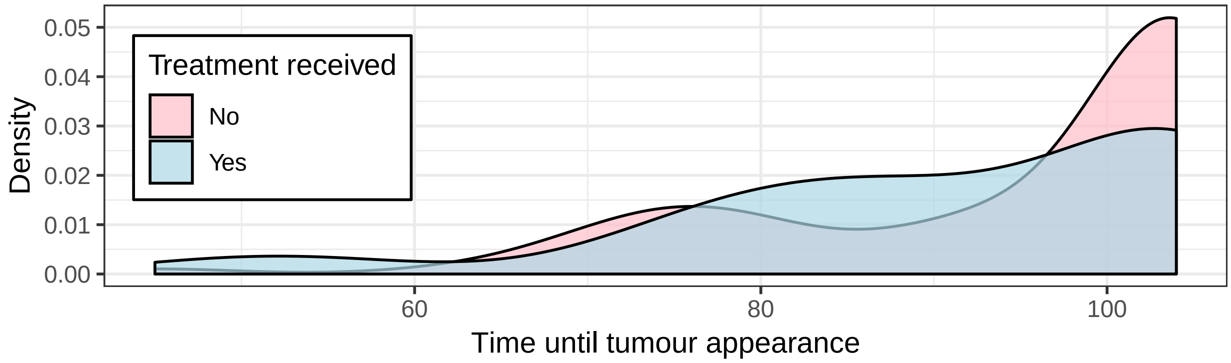

Figure 1: Density plots for uncensored observations

```
# Log-probability density
weib_prob <- deriv(
  expression(
    -log_sigma - (beta0 + beta1*treated) +
      ((1/exp(log_sigma))-1)*(log(t) - (beta0 + beta1*treated)) -
      (t/exp(beta0 + beta1*treated))^(1/exp(log_sigma))
  ),
  namevec = c("beta0", "beta1", "log_sigma"),
  function.arg = c("t", "beta0", "beta1", "log_sigma", "treated"),
  hessian = TRUE
)


# Log-survival
weib_surv <- deriv(
  expression(
    -(t/exp(beta0 + beta1*treated))^(1/exp(log_sigma))
  ),
  namevec = c("beta0", "beta1", "log_sigma"),
  function.arg = c("t", "beta0", "beta1", "log_sigma", "treated"),
  hessian = TRUE
)


# Negative log-likelihood
weib_nll <- function(theta, t, treated, status) {

  beta0 <- theta[1]
  beta1 <- theta[2]
  log_sigma <- theta[3]

  # Use pdf for uncensored observations, and survival function for censored
  -sum(
    status * weib_prob(t, beta0, beta1, log_sigma, treated),
    (1-status) * weib_surv(t, beta0, beta1, log_sigma, treated)
  )
}


# Gradient of negative log-likelihood
weib_nll_gr <- function(theta, t, treated, status) {

  beta0 <- theta[1]
  beta1 <- theta[2]
  log_sigma <- theta[3]

  -colSums(rbind(
    status * attr(weib_prob(t, beta0, beta1, log_sigma, treated), "gradient"),
    (1-status) * attr(weib_surv(t, beta0, beta1, log_sigma, treated), "gradient")
  ))
}
```

By minimising the negative log-likelihood, we obtain a maximum likelihood estimate for the parameters.

```
# Initial parameters
theta0 <- c("beta0" = 2, "beta1" = 2, "log_sigma" = 2)

# Minimise negative log-likelihood
weib_opt <- optim(par = theta0, fn = weib_nll, gr = weib_nll_gr,
                  t = rats$time, treated = rats$rx, status = rats$status,
                  method = "BFGS", hessian = TRUE)

# If convergence was achieved, print optimised parameters
if (weib_opt$convergence != 0) {
  message("Optimisation did not converge")
} else {
  weib_opt$par
}
```

```
##      beta0      beta1  log_sigma
##  4.9831358 -0.2385112 -1.3326112
```

By this process we have obtained parameter values of $\beta_0 = 4.983$, $\beta_1 =$ -0.239 and $\sigma = 0.264$.

Note that we have asked `optim()` to return the Hessian matrix evaluated at the optimised parameter values. In fact, since we have minimised the *negative* log-likelihood, we have been given the negative Hessian; inverting this yields an approximation to the (asymptotic) covariance matrix, and therefore the standard errors for each parameter can be estimated by taking the square roots of the elements on the main diagonal.

```
weib_std_err <- sqrt(diag(solve(weib_opt$hessian)))
weib_std_err
```

```
##      beta0      beta1  log_sigma 
## 0.08332142 0.08908413 0.14387892
```

Section 1.2: Confidence intervals

With the standard errors obtained at the end of the previous section, we can calculate asymptotic confidence intervals for each of the three parameters, since we know that the distributions of maximum likelihood estimates are asymptotically normal. For example, the 95% asymptotic confidence interval for β_1 can be calculated as follows:

```
weib_beta1_ci <- weib_opt$par["beta1"] +
  c("lower" = -1, "upper" = 1)*qnorm(0.975)*weib_std_err["beta1"]
weib_beta1_ci
```

```
##       lower       upper 
## -0.41311289 -0.06390951
```

This interval provides strong evidence that β_1 is negative, so let us take a moment to interpret this in the context of our analysis.

The Weibull model is influenced by β_1 in terms of η_i: if the ith rat has recieved treatment, then $\eta_i = \beta_0 + \beta_1$, otherwise $\eta_i = \beta_0$. Therefore if β_1 is negative, then η_i is lower for treated rats. Then the scale parameter in the Weibull distribution is also smaller, since we have used scale $= \exp(\eta_i)$ and the exponential is an increasing function.

The analytical mean of a Weibull distribution with shape a and scale b can be calculated as $b\Gamma(1 + 1/a)$, where Γ is the Gamma function. Hence using our MLE parameters, we can calculate estimates for the expected means of the treated and untreated observations. Code for this is included in the accompanying R Markdown file. Values for the means with different status' and β_1 values are as follows:

```
##          treated treated.ci.lower treated.ci.upper        untreated
##         103.8897          87.2457         123.7090         131.8732
```

Notice that there is a difference of around 30 weeks in tumour appearance between the estimated means of the two groups - in fact, the estimated mean for the untreated population lies well outside the estimated 95% confidence interval for the mean of the treated population. This provides very strong evidence in favour of the hypothesis we tentatively proposed after our quick initial inspection of the data; that is, it suggests that rats which had received treatment developed tumours earlier than their untreated counterparts.

Section 1.3: Considering an alternative distribution

Our initial decision to model the data using a Weibull distribution was somewhat arbitrary. Therefore in order to gain a different perspective, we now repeat the processes followed in the previous sections, but instead assuming that T_i can be modelled by a log-logistic distribution (and its corresponding survival function).

The log-logistic distribution is another popular option for survival analysis, and is also parameterised by shape and scale. We express these two parameters in terms of β_0, β_1 and σ in exactly the same way as we did for the Weibull model; and by defining nearly identical R functions using `deriv()` to those defined in Section 1.1, we can calculate maximum likelihood estimates for each of the three parameters.

```
# Minimise negative log-likelihood
llog_opt <- optim(par = theta0, fn = llog_nll, gr = llog_nll_gr,
                  t = rats$time, treated = rats$rx, status = rats$status,
                  method = "BFGS", hessian = TRUE)

if (llog_opt$convergence != 0) {
  message("Optimisation did not converge")
} else {
  llog_opt$par
}
```

```
##     beta0     beta1 log_sigma 
##  4.916652 -0.229599 -1.409816
```

Our parameter estimates are therefore $\beta_0 = 4.917$, $\beta_1 =$ -0.23 and $\sigma = 0.244$; notice that although we have assumed a different underlying distribution, these values are remarkably similar to those obtained in Section 1.1.

An asymptotic 95% confidence interval for β_1 can be calculated in the same way as in Section 1.2, yielding (-0.417, -0.042); and using that the analytical mean of a log-logistic distribution with shape a and scale b can be calculated as $\frac{a\pi/b}{\sin(\pi/b)}$, just as in Section 1.2 we can calculate estimations of the mean tumour appearance times of the treated and untreated populations.

```
##          treated treated.ci.lower treated.ci.upper          untreated
##         130.4172         108.1233         157.3078           164.0773
```

Again, there is a large difference between the estimated means, and the mean of the untreated population lies far outside the confidence interval of the mean of the treated population. Once again, this strongly suggests that rats which received treatment developed tumours earlier than those which received a placebo.

Section 1.4: Incorporating litter as a random effect

We now update the model to include the effect of litters as random effects. Each rat belonged to a particular litter of three rats total: some litters may have been susceptible to developing tumours earlier or later than other litters, so modelling the litter as a random effect allows that to be accounted for.

Reverting to our original Weibull model from earlier sections, we redefine $\eta_i = \beta_0 + \beta_1 x_i + b_{litter(i)}$ to include a term for the random effect induced by the ith litter, and construct new log-PDF and log-survival functions in terms of this η_i:

```
weib_re_prob <- deriv(
  expression(
    -log_sigma - eta + (1/exp(log_sigma) - 1)*(log(t) - eta) -
      (t/exp(eta))^(1/exp(log_sigma))
  ),
  namevec = "eta",
  function.arg = c("t", "eta", "log_sigma"),
  hessian = TRUE
)

weib_re_surv <- deriv(
  expression(
    -(t/exp(eta))^(1/exp(log_sigma))
  ),
  namevec = "eta",
  function.arg = c("t", "eta", "log_sigma"),
  hessian = TRUE
)
```

We now write a function which will calculate the log-joint probability density of a set of observations and a corresponding set of random effects. At this point, it is conceptually useful to define some inputs which will later be passed into this function. Specifically, we consider two model matrices: one for the deterministic parameters β_0 and β_1, which we will call `X`, and another for the random effects, which we will call `Z`.

```
# Log-likelihood function l(y) = l(y, b)
lfyb <- function(theta, y, b, X, Z) {

  beta <- theta[1:2]
  log_sigma <- theta[3]
  log_sigma_b <- theta[4]

  eta <- X%*%beta + Z%*%b

  time <- y[, "time"]
  status <- y[, "status"]

  # Log conditional density of y given b
  lfy_b <- sum(status * weib_re_prob(time, eta, log_sigma),
               (1-status) * weib_re_surv(time, eta, log_sigma))

  # Log marginal density of b
  lfb <- sum(dnorm(x = b, mean = 0, sd = exp(log_sigma_b), log = TRUE))

  # Log joint density of y and b is the sum of independend log densities
  lf <- lfy_b + lfb

  # Gradient of log-likelihood with respect to b
  # 2 terms: d(lfy_b)/db + d(lfb)/db
  g <- t(Z) %*%
    (status*attr(weib_re_prob(time, eta, log_sigma), "gradient") +
       (1-status)*attr(weib_re_surv(time, eta, log_sigma), "gradient")) -
    b/(exp(log_sigma_b)^2)

  # Hessian of log-likelihood wrt b, i.e. dl/(db db^T)), will be diagonal since
  # taking derivative wrt b_i then wrt b_j where j!=i gives us 0
  # i.e. we can use "hessian" from weib_re_*()
  # So H_diag is the diagonal entries from the Hessian (all other entries are 0)
  H_diag <- t(Z) %*%
    (status*attr(weib_re_prob(time, eta, log_sigma), "hessian") +
       (1-status)*attr(weib_re_surv(time, eta, log_sigma), "hessian")) -
    1/(exp(log_sigma_b)^2)

  list(lf = lf, g = g, H_diag = H_diag)
}
```

In order to obtain the marginal negative log-likelihood of our parameter vector θ using this log-likelihood function, it is necessary to integrate over the random effects **b**. This is not an easy integral to solve analytically; so we can calculate an approximate value using the Laplace approximation.

```
# Laplace approximation calculation of marginal negative log-likelihood of theta
lal <- function(theta, y, X, Z) {

  opt <- optim(
    par = 0.01*rnorm(ncol(Z)), # initial guess for b
    fn = function(b) lfyb(theta, y, b, X, Z)$lf,
    gr = function(b) lfyb(theta, y, b, X, Z)$g,
    # Set `fnscale = -1` so that we MINIMISE the NEGATIVE log-likelihood
    method = "BFGS", control = list(fnscale = -1)
  )

  soln_opt <- lfyb(theta, y, opt$par, X, Z)

  # Hessian was diagonal, so det(-H) is product of diagonal elements
  # i.e. log(det(-H)) is sum of logs of diagonal elements
  # We use abs() since sometimes very small values have the wrong sign
  # (due to floating point errors)
  log_det_H <- sum(log(abs(soln_opt$H_diag)))

  # Return (logged) Laplace approximation of negative log-likelihood
  -((length(opt$par)/2)*log(2*pi) + soln_opt$lf - log_det_H/2)
}
```

Having defined these two functions, we can minimise the marginal negative log-likelihood using `optim()`. We use BFGS, although we do not provide a gradient function; finite-difference approximation is therefore used to determine the gradient at each iteration, and consequently the optimisation is rather unstable. The choice of starting values is therefore much more important here than it was in previous sections.

```
# Starting values, selected empirically
theta_init <- c("beta0" = 5, "beta1" = -1, "log_sigma" = -1, "log_sigma_b" = -2)

re_opt <- optim(theta_init, fn = lal, y = rats,
                X = model.matrix(~ 1 + rx, data = rats),
                Z = model.matrix(~ litter - 1, data = rats),
                method = "BFGS", hessian = TRUE)

if (re_opt$convergence != 0) {
  message("Optimisation did not converge")
} else {
  re_opt$par
}
```

```
##       beta0       beta1   log_sigma log_sigma_b
##   5.0077257  -0.2316954  -1.3777672  -1.6784099
```

As before, we use the returned Hessian to find estimated standard errors for each parameter, and to calculate an asymptotic 95% confidence interval for β_1:

```
re_std_err <- sqrt(diag(solve(re_opt$hessian)))

re_beta1_ci <- re_opt$par["beta1"] +
  c("lower" = -1, "upper" = 1)*qnorm(0.975)*re_std_err["beta1"]
re_beta1_ci
```

```
##      lower      upper 
## -0.2755764 -0.1878144
```

Compare this confidence interval with the one calculated in Section 1.2: the confidence interval is significantly tighter when random effects are taken into account. Therefore the inclusion of random effects in our model has improved our parameter estimates and shows increasing evidence that the use of this treatment is causing earlier tumor appearance in treated rats. Additionally we can infer that the litter does indeed have an effect on the time it takes for tumours to appear in rats.

Section 1.5: Bayesian sampling of parameters

We now use the same model as in the previous section, including random effects, and create a Bayesian MCMC sampling procedure based on the Metropolis-Hastings algorithm to estimate unknown parameter values.

Firstly we define the log of the posterior distribution to be used for MCMC, using improper uniform priors for β_0, β_1 and $\log(\sigma)$, and an exponential prior with rate 5 for σ_b.

```r
log_posterior <- function(theta, y, b, X, Z) {
  
  # Log likelihood of y and b
  lf <- lfyb(theta, y, b, X, Z)$lf
  
  # Define log-prior, sum of log-priors for all parameters
  # Here, log-prior is 0 for all but log_sigma_b
  log_prior <- dexp(exp(theta[4]), rate = 5, log = TRUE)
  
  # Log-posterior is log-prior plus log-likelihood
  lf + log_prior
}
```

Now we construct a function which performs the Metropolis-Hastings algorithm, updating the "non-random" and "random" parameters independently at each iteration.

```r
mcmc_mh <- function(iters, burnin, init_params, init_bs, tuners, b_tuner,
                    y, X, Z, show_plot = TRUE) {
  
  theta_vals <- matrix(NA, nrow = iters+1, ncol = length(init_params))
  colnames(theta_vals) <- names(init_params)
  theta_vals[1, ] <- init_params
  
  b_vals <- matrix(NA, nrow = iters+1, ncol = length(init_bs))
  b_vals[1, ] <- init_bs
  
  acceptance <- rep(NA, iters)
  acceptance_b <- rep(NA, iters)
  
  log_post <- log_posterior(init_params, y, init_bs, X, Z)
  
  # MCMC loop
  for (k in seq_len(iters)) {
    
    # "Non-random" parameters
    theta_prop <- rnorm(length(init_params), mean = theta_vals[k, ], sd = tuners)
    log_post_prop <- log_posterior(theta_prop, y, b_vals[k, ], X, Z)
    accept_prob <- exp(log_post_prop - log_post)
    
    if (accept_prob > runif(1)) {
      theta_vals[k+1, ] <- theta_prop
      log_post <- log_post_prop
      acceptance[k] <- TRUE
    } else {
      
      theta_vals[k+1, ] <- theta_vals[k, ]
      acceptance[k] <- FALSE
    }
    
    # "Random" parameters
    b_prop <- rnorm(length(init_bs), mean = b_vals[k, ], sd = b_tuner)
    
    # Use (possibly newly accepted) values of theta
    log_post_prop_b <- log_posterior(theta_vals[k+1, ], y, b_prop, X, Z)
    accept_prob_b <- exp(log_post_prop_b - log_post)
```

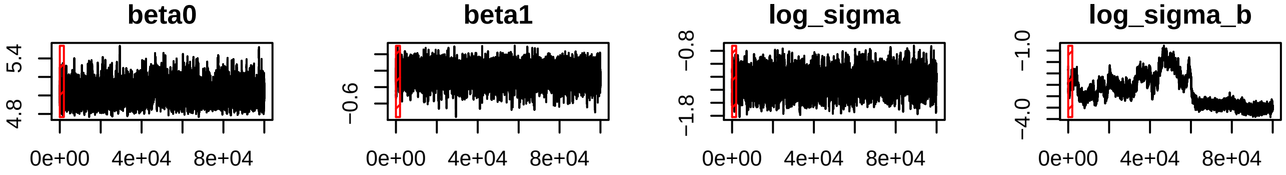

Figure 2: Samples from Metropolis-Hastings MCMC

```
    if (accept_prob_b > runif(1)) {
      b_vals[k+1, ] <- b_prop
      log_post <- log_post_prop_b
      acceptance_b[k] <- TRUE
    } else {
      b_vals[k+1, ] <- b_vals[k, ]
      acceptance_b[k] <- FALSE
    }
  }

  if (show_plot) {

    par(mfrow = c(1, ncol(theta_vals)), mai = rep(0.3, 4))
    lapply(names(init_params), function(nm) {
      y <- theta_vals[, nm]
      plot(y, type = "l", main = nm, xlab = "Iteration", ylab = "")
      rect(0, min(y), burnin, max(y), density = 10, col = "red")
    })
    par(mfrow = c(1, 1))
  }

  cat(sprintf("Total acceptance:       %2.3f%%\n", mean(acceptance)*100))
  cat(sprintf("Burned-in acceptance:   %2.3f%%\n", mean(acceptance[-(1:burnin)])*100))
  cat(sprintf("Burned-in b acceptance: %2.3f%%\n", mean(acceptance_b[-(1:burnin)])*100))

  list(theta = theta_vals, b = b_vals)
}
```

With this function defined, we can easily run 100000 iterations of MCMC. We use the MLE parameter values from Section 1.4 as starting values since we know these should already be reasonable parameter estimates. Tuning parameters are 0.1 for all parameters, and 0.03 for the random effects (which are initialised at 0) - empirically, these provide good acceptance probabilities for both sets of parameters. We also define a burn-in period of 2000 iterations, which allows the MCMC chains to reach the regions where they produce good random samples.

```
iters <- 100000
burnin <- 2000

pilot <- mcmc_mh(
  iters, burnin, re_opt$par, rep(0, 50), c(0.1, 0.1, 0.1, 0.1), 0.03,
  y = rats[, c("time", "status")],
  X = model.matrix(~ 1 + rx, data = rats),
  Z = model.matrix(~ litter - 1, data = rats)
)
```

```
## Total acceptance:       24.216%
## Burned-in acceptance:   24.237%
## Burned-in b acceptance: 15.274%
```

Discarding the values generated during the burn-in period, we can check for correlation between the “non-random” parameters in θ.

```
cor(pilot$theta[-(1:burnin), ])
```

```
##                   beta0      beta1  log_sigma log_sigma_b
## beta0        1.00000000 -0.7144942  0.6866077  0.05000082
## beta1       -0.71449423  1.0000000 -0.3744299  0.05836160
## log_sigma    0.68660765 -0.3744299  1.0000000 -0.13576550
## log_sigma_b  0.05000082  0.0583616 -0.1357655  1.00000000
```

This reveals that there is significant correlation between parameters, e.g. strong negative correlation between β_0 and β_1.

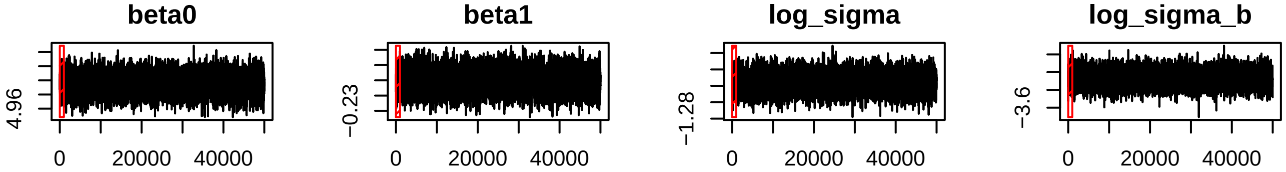

Figure 3: Samples from correlation-adjusted MCMC

Consequently, we shall now adapt our MCMC approach by attempting to account for the correlations. At each step, the proposed θ and $\mathbf{b}$ are generated using a multivariate normal proposal distribution, with covariance matrix `cov(D)` (where `D` is the matrix of combined MCMC results for θ and $\mathbf{b}$, minus the burn-in period). We use the last values generated by the above MCMC algorithm as starting values in the new algorithm, and new proposed values are centred around these.

This is implemented in a new function called `mcmc_mh_cov()`, which takes a single tuning parameter and the covariance matrix `cov(D)` in place of the 5 tuning parameters from before. The function is identical to `mcmc_mh()` other than the structure of the MCMC loop, where new values for θ and $\mathbf{b}$ are proposed and accepted or rejected simultaneously:

```r
props <- MASS::mvrnorm(iters, mu = c(init_params, init_bs), Sigma = tuner^2 * cov_matrix)

# MCMC loop in mcmc_mh_cov()
for (k in seq_len(iters)) {

  theta_prop <- props[k, 1:length(init_params)]
  b_prop <- props[k, (length(init_params)+1):ncol(props)]
  log_post_prop <- log_posterior(theta_prop, y, b_prop, X, Z)
  accept_prob <- exp(log_post_prop - log_post)

  if (accept_prob > runif(1)) {
    theta_vals[k+1, ] <- theta_prop
    b_vals[k+1, ] <- b_prop
    log_post <- log_post_prop
    acceptance[k] <- TRUE
  } else {
    theta_vals[k+1, ] <- theta_vals[k, ]
    b_vals[k+1, ] <- b_vals[k, ]
    acceptance[k] <- FALSE
  }
}
```

Note that the single tuning parameter now controls all our proposed values simultaneously.

Running this new sampler over 50000 iterations and a smaller burn-in period of 1000 (reflecting our greater confidence in our starting values), the plots show good convergence to a stationary distribution, visible as the resemblance of the sampled values to white noise.

```r
D <- cbind(pilot$theta, pilot$b)[-(1:burnin), ]

iters <- 50000
burnin <- 1000

adjusted <- mcmc_mh_cov(
  iters, burnin, drop(tail(pilot$theta, 1)), drop(tail(pilot$b, 1)), cov(D), 0.07,
  y = rats[, c("time", "status")],
  X = model.matrix(~ 1 + rx, data = rats),
  Z = model.matrix(~ litter - 1, data = rats)
)
```

```
## Total acceptance:       23.182%
## Burned-in acceptance:   23.147%
```

Combining the newly obtained values of θ and $\mathbf{b}$, we check their correlations and the appearance of their marginal densities. In Figure 4, marginal densities are shown on the diagonal, correlation plots between parameters to the left of the diagonal, and calculated correlation values to the right.

Note that some of these parameters are still correlated. This is not something we have changed (indeed, it is not something we are *able* to change); but we have now accounted for this correlation in our sampling.

As a final step, we run a Kolmogorov-Smirnov test on a sample from each of the parameters to provide evidence for the fact that we have converged to the stationary distribution.

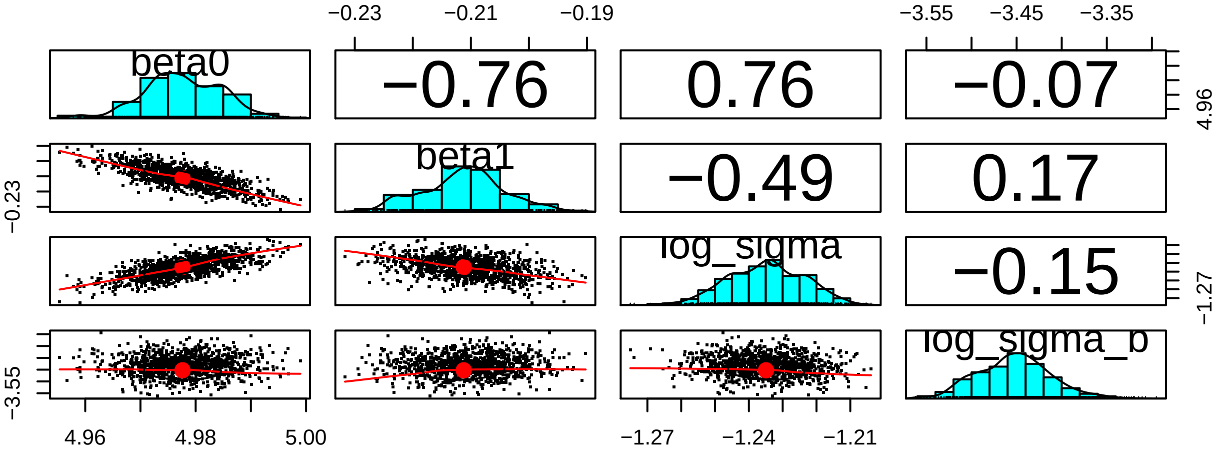

Figure 4: Correlation between parameters

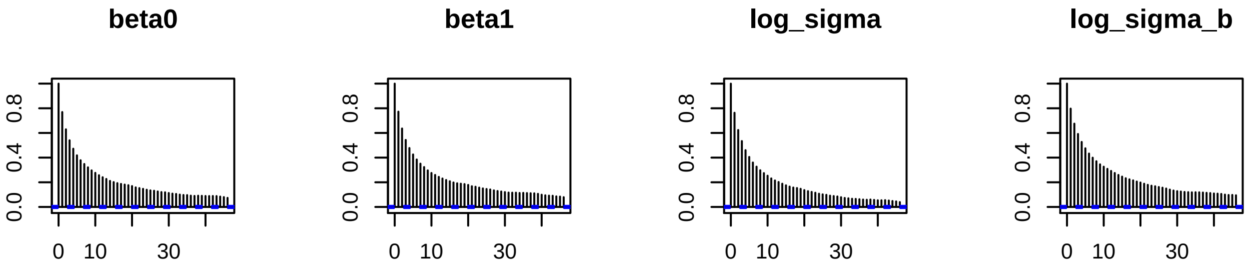

Figure 5: Autocorrelation of parameters

We first calculate the autocorrelation values, and from there calculate the “autocorrelation length” (ACL), defined as the distance between two effectively-independent observations. We then take a sample of the parameter, taking every ACLth value, and thereby obtaining a sample which can be assumed to consist of independent observations. We then calculate two further subsamples of this sample, and run the K-S test on these. The `ks.test()` function in R warns us loudly that if ties are present in our data, it cannot calculate an exact p-value; we acknowledge this, but deem it not to be a problem due to our large sample size.

```r
par(mfrow = c(1, 4), mai = rep(0.4, 4))

ks_pvals <- suppressWarnings({vapply(colnames(adjusted$theta), function(nm) {

    # Calculate autocorrelation in MH sample for parameter
    y <- adjusted$theta[-(1:burnin), nm]
    autocorr <- acf(y, main = nm)

    # Sample of ACL-spaced observations
    acl <- 2*sum(autocorr$acf) + 1
    ac_smp <- y[seq(from = 1, to = length(y), by = acl)]

    # Test whether two halves of this sample are from same distribution
    idx <- sample(1:length(ac_smp), ceiling(length(ac_smp)/2), replace = FALSE)
    ks.test(ac_smp[idx], ac_smp[-idx])$p.value

  }, FUN.VALUE = 0)})
```

```r
par(mfrow = c(1, 1))
ks_pvals
```

```
##       beta0       beta1   log_sigma log_sigma_b
##   0.9832580   0.5912364   0.7913256   0.7037153
```

The p-values are all above 0.05, so at the 5% significance level there is no evidence to suggest that the two subsamples have come from different distributions. We can therefore assume we have converged to the stationary distribution.

Satisfied with our sampling, we conclude by calculating a 95% credible interval for β_1.

```r
quantile(adjusted$theta[-(1:burnin), "beta1"], c(0.025, 0.975))
```

```
##       2.5%      97.5%
## -0.2236685 -0.1967210
```

Carcinogenesis Study on Rats: Conclusion

We have used mutliple determenistic models, included random effects and carried out a Bayesian estimation procedure to analyse whether there is any evidence that the drug in question is carcinogenic to rats. All our models point towards a lower β_1 value in treated rats, and based on our models this would imply that the use of this drug is causing earlier tumour appearance in treated rats and hence is carcinogenic.

Part 2: Fatigue of Materials

The data used in this section can be downloaded from the following location:

```r
fatigue <- read.table("http://people.bath.ac.uk/kai21/ASI/fatigue.txt")
set.seed(35647663) # Set a random seed for reproducibility
```

Section 2.1: Maximum likelihood estimation of distribution parameters

In this section we investigate fatigue in material samples subjected to cyclic loading. Conceptually, this problem is very similar to the problem investigated in Part 1: that is, the Weibull distribution is at the heart of much of the following analysis, and observations are treated as censored or uncensored based on whether or not the fatigue-testing run was completed.

We have assumed that N_i, the number of test cycles completed at stress level s_i and with fatigue constant γ for the ith observation, is given by

$$N_i = \alpha(s_i - \gamma)^\delta \epsilon_i$$

where

$$\epsilon_i \sim \text{Weibull}\,(\text{shape} = 1/\sigma,\ \text{scale} = 1)$$

Immediately we can incorporate the rest of the formula for N_i into the scale parameter of this distribution, obtaining that

$$N_i \sim \text{Weibull}\left(\text{shape} = 1/\sigma,\ \text{scale} = \alpha(s_i - \gamma)^\delta\right)$$

In the same way as in Section 1.1, we derive expressions for the negative log-likelihood and its gradient with respect to each of the three parameters, making use of R's `deriv()` function. Within these we reparameterise α and σ using a log transform, since both of these parameters must always be greater than 0. Having defined these functions, we can select a value for γ and then optimise over θ.

```r
# Initial params
theta0 <- c("log_alpha" = 2, "delta" = 1, "log_sigma" = 2)
gamma <- 70

# Minimise negative log-likelihood
N_opt_const_gamma <- optim(
  par = theta0, fn = N_nll_const_gamma, gr = N_nll_gr_const_gamma,
  y = fatigue$N, gamma = gamma, stress = fatigue$s, runout = fatigue$ro,
  method = "BFGS", hessian = TRUE
)
```

Again, just as in Part 1, we can use the negative Hessian returned by `optim()` to calculate standard errors and hence 95% asymptotic confidence intervals for each parameter.

```r
# Calculate standard errors from inverse Hessian
N_std_err_const_gamma <- sqrt(diag(solve(N_opt_const_gamma$hessian)))

# 95% confidence interval for each parameter (asymptotic distribution is normal)
round(data.frame(
  val = N_opt_const_gamma$par,
  se = N_std_err_const_gamma,
```

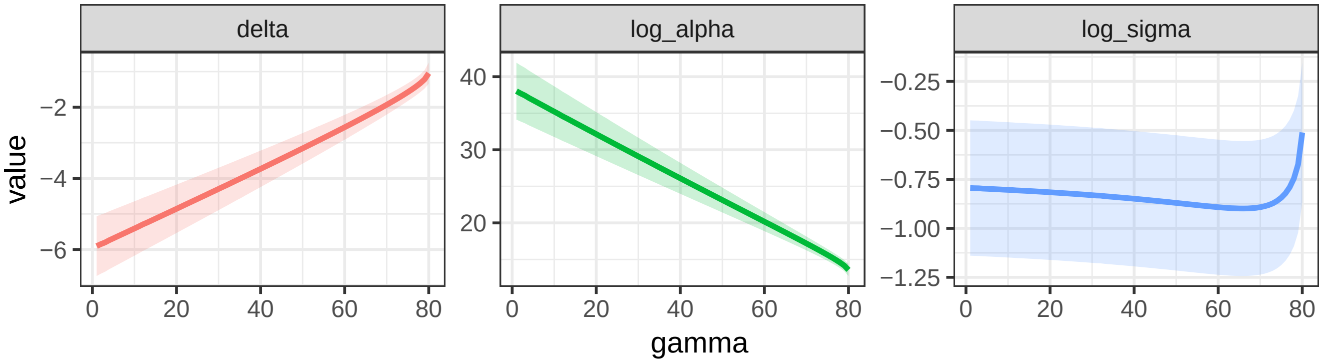

Figure 6: Optimised parameters across the range of potential fatigue limits

```
  lower = N_opt_const_gamma$par - qnorm(0.975)*N_std_err_const_gamma,
  upper = N_opt_const_gamma$par + qnorm(0.975)*N_std_err_const_gamma
), 3)
```

```
##              val    se  lower  upper
## log_alpha 17.190 0.484 16.242 18.138
## delta     -1.932 0.141 -2.208 -1.656
## log_sigma -0.892 0.176 -1.236 -0.548
```

The optimisation is of course dependent on our chosen value of the fatigue limit γ. By carrying out similar optimisations for different values of γ across its potential range, we can see in Figure 6 that the parameter values change considerably. In particular, note the apparent negative correlation between `log_alpha` and `delta` (and the steadily tightening 95% confidence intervals for these paramaters); and the rather more dramatic curve that `log_sigma` takes towards the upper end of the range of γ (although the confidence interval seems to be of more or less constant width across the whole range).

Section 2.2: Estimation of fatigue limit

Theoretically, rather than selecting γ ourselves it is possible to include it in the optimisation. The only slight technicality is that we must transform γ from its current bounded interval - specifically, $\gamma \in (0, \min(s_i))$ - onto an unbounded interval. This is achievable via the logit transformation, i.e. we optimise using

$$\text{l\_gamma} = \text{logit}\left(\frac{\gamma}{\min(s_i)}\right) = \log\left(\frac{\gamma}{\min(s_i) - \gamma}\right)$$

The code from Section 2.1 can be adapted easily by replacing the constant `gamma` in our expressions with `min_s/(1 + exp(-l_gamma))` and by adding `l_gamma` to the list of parameters for which we would like to return a gradient. Optimisation can then be performed as in Section 2.1.

```
# Initial parameters, selected empirically
theta0 <- c("log_alpha" = 2, "delta" = 1, "log_sigma" = 2, "l_gamma" = 0)

# Minimise negative log-likelihood
N_opt <- optim(theta0, fn = N_nll, gr = N_nll_gr,
  y = fatigue$N, stress = fatigue$s, runout = fatigue$ro,
  method = "BFGS", hessian = TRUE
)
```

We can then calculate confidence intervals for these parameters in exactly the same way as before, using the negative Hessian returned by the optimisation.

```
##              val    se  lower  upper
## log_alpha 18.255 2.690 12.984 23.527
## delta     -2.163 0.583 -3.305 -1.021
## log_sigma -0.899 0.176 -1.243 -0.554
## l_gamma    1.570 0.770  0.061  3.078
```

The optimised value of γ can be retrieved using the inverse-logit (sigmoid) transformation:

```
min(fatigue$s) / (1 + exp(-N_opt$par[4]))
```

```
##  l_gamma
## 66.46811
```

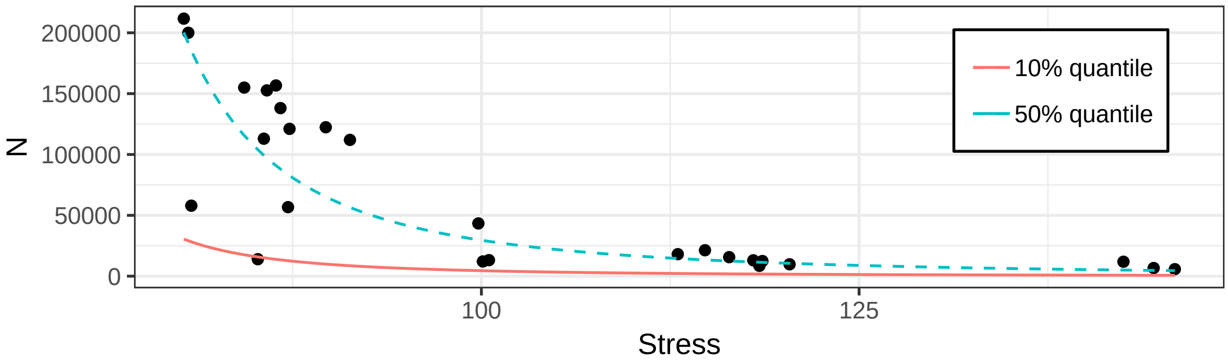

Figure 7: Quantiles of our approximated distribution of N

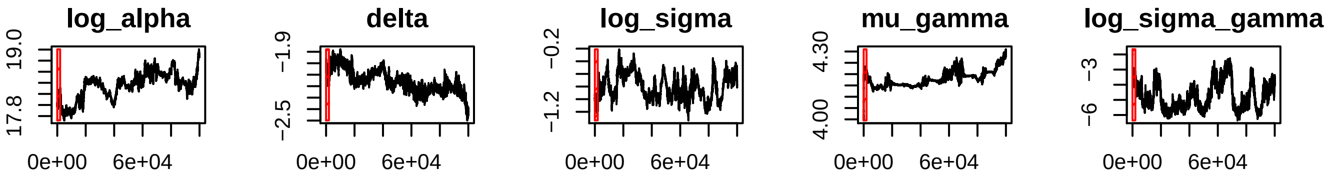

Figure 8: MCMC samples for the 5 fatigue parameters

Interestingly, this "optimal" γ seems to lie just before the upward turn seen in Figure 6, providing further evidence that the relationship between σ and γ is the most important factor contributing to the distribution of N_i.

Section 2.3: Important quantiles of approximated distribution

Using the optimised parameters from the previous section, we can compute approximate 10% and 50% quantiles for the distribution of N over the range of stress values used in the fatigue testing. These are shown, alongside the recorded observations, in Figure 7; note that all but one of our observations lie clearly above the 10% quantile.

Section 2.4: Modelling fatigue limit as a random effect

We now apply a Bayesian approach to the above random effects model, and again create a Bayesian MCMC sampling procedure based on the Metropolis-Hastings algorithm to estimate unknown parameter values.

We define the log of the posterior distribution to be used for MCMC in a similar way to Section 1.4, using improper uniform priors for $log(\alpha)$, δ, $\log(\sigma)$ and $log(\mu_\gamma)$ and an exponential prior with rate 5 for $log(\sigma_\gamma)$. We assume all of the priors are independent. We then apply the Metropolis Hastings algorithm using independent normal proposal distributions for all parameters and random effects, centred around the previous accepted values. We use the same Metropolis Hastings algorithm from Section 1.5 here.

```
# Initial paramteers, selected empirically
theta0 <- c(log_alpha = 18, delta = -2, log_sigma = -1,
            mu_gamma = 4, log_sigma_gamma = -2)

iters <- 100000
burnin <- 2000

# Initial gamma values based on optimal gamma from Section 2.2
pilot <- mcmc_mh(
  iters, burnin, theta0, rep(66, 26), c(0.01, 0.01, 0.015, 0.01, 0.13), 0.25,
  y = fatigue$N, X = fatigue$s, Z = fatigue$ro
)
```

```
## Total acceptance:       20.439%
## Burned-in acceptance:   19.988%
## Burned-in b acceptance: 29.315%
```

From the trace plots alone (see Figure 8) it is clear that again there is significant correlation between parameters. Therefore we apply the same approach as in Section 1.5, whereby account for this correlation and run the alternate version of the Metropolis Hastings algorithm. Once again, we run 50000 iterations and define a short burn-in period of 1000 iterations to allow the chains (initialised at the final values from the pilot run) to fully stabilise.

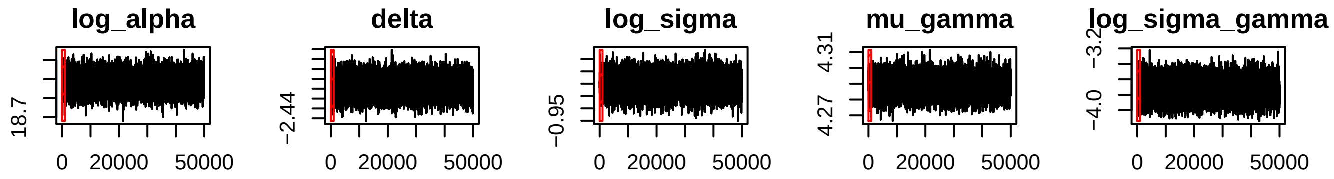

Figure 9: Fatigue parameter values obtained with covariance-adjusted MCMC sampling

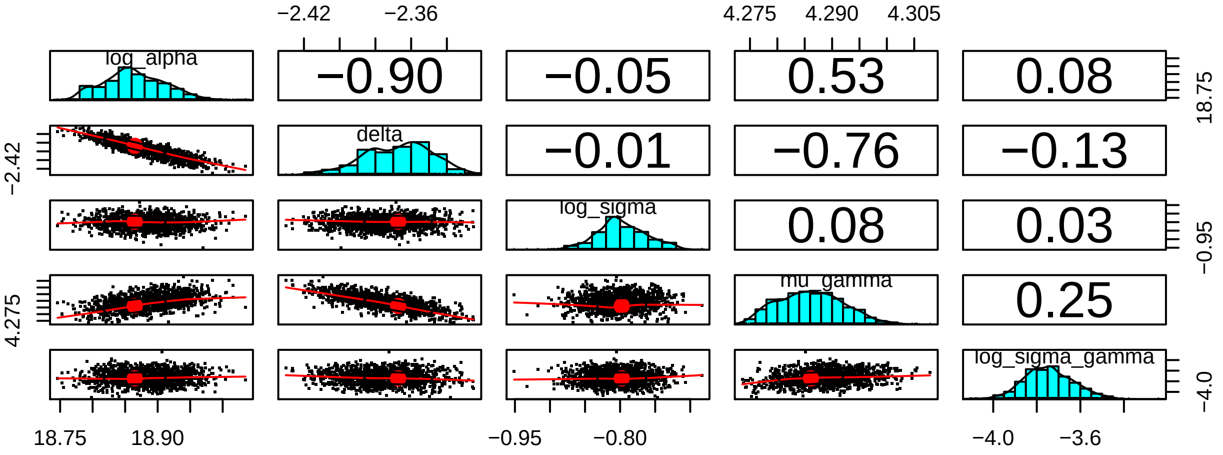

Figure 10: Correlation between fatigue parameters

```
D <- cbind(pilot$theta, pilot$b)[(burnin+1):iters, ]

adjusted <- mcmc_mh_cov(
  50000, 1000, drop(tail(pilot$theta, 1)), drop(tail(pilot$b, 1)), cov(D), 0.2,
  y = fatigue$N, X = fatigue$s, Z = fatigue$ro
)
```

```
## Total acceptance:       26.626%
## Burned-in acceptance:   26.594%
```

Finding parameter marginal distributions and viewing correlations, as before:

We now again check convergence to the stationary distribution using a Kolmogorov-Smirnov test.

```
ks_pvals_f
```

```
##       log_alpha          delta       log_sigma        mu_gamma
##       0.3043200      0.9818910       0.1862747       0.9574697
```

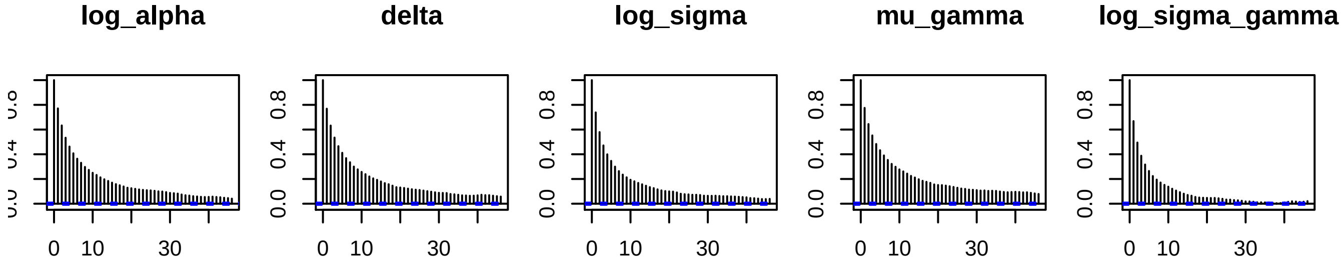

Figure 11: Autocorrelation of fatigue parameters

```
## log_sigma_gamma
##       0.9428152
```

All p-values of the K-S test are above 0.05, so at the 5% significance level there is no evidence to suggest that the two subsamples come from different distributions. Therefore there is no evidence to reject the claim that we have indeed converged to the stationary distribution.

Satisfied with our sampling, we conclude by calculating 95% credible intervals for the parameters.

```r
vapply(colnames(adjusted$theta), function(nm) {
  quantile(adjusted$theta[-(1:burnin), nm], c(0.025, 0.975))
}, FUN.VALUE = c(0, 0))
```

```
##       log_alpha     delta  log_sigma mu_gamma log_sigma_gamma
## 2.5%   18.77929 -2.404552 -0.8707027 4.275086       -3.953575
## 97.5%  18.95540 -2.332513 -0.7238820 4.297904       -3.511619
```

Section 2.5: Comparison with numerical integration

The above approach excluded the denominator of the posterior distribution, namely $f(\mathbf{n})$. We assume each n_i is independent, for $i = 1, ..., 26$, and therefore we have:

$$f(\mathbf{n}) = \prod_{i=1}^{26} f(n_i) = \prod_{i=1}^{26} \int_0^{s_i} f(n_i|\gamma_i) f(\gamma_i) d\gamma_i$$

The function `intgrl()` calculates an approximation to the (intractable) integral for each n_i, given $\theta^\top = (\log(\alpha), \delta, \log(\sigma), \mu_\gamma, \log(\sigma_\gamma))$.

```r
intgrl <- function(theta) {

  log_alpha <- theta[1]
  delta <- theta[2]
  log_sigma <- theta[3]
  mu_gamma <- theta[4]
  log_sigma_gamma <- theta[5]

  vapply(seq_len(nrow(fatigue)), function(k) {
    integrate(
      function(gamma) {

        shape <- 1/exp(log_sigma)
        scale <- (exp(log_alpha)*(fatigue$s[k] - gamma)^delta)

        ((1-fatigue$ro[k]) * dweibull(fatigue$N[k], shape = shape, scale = scale) +
           fatigue$ro[k] * pweibull(fatigue$N[k], shape = shape, scale = scale,
                                    lower.tail = FALSE)) *
          dweibull(gamma, shape = 1/exp(log_sigma_gamma), scale = exp(mu_gamma))
      },
      lower = 0, upper = fatigue$s[k]
    )$value
  }, FUN.VALUE = 0)
}
```

We can now calculate an approximation to the full log posterior equation, utilising the above function. We use improper uniform priors for all parameters in this case (the auxiliary function `log_post_imp_priors()` calculates the log-likelihood assuming this).

```r
# Create full log posterior distribution with Bayesian denominator included
log_posterior <- function(theta, y, b, stress, runout) {
  lf_int <- sum(log(intgrl(theta)))
  lf <- log_post_imp_priors(theta, y, b, stress, runout)
  lf - lf_int
}
```

We now do an MCMC run using the above `log_posterior()` function and calculate 95% credible intervals for each parameter, to compare with Section 2.4.

```r
# MCMC again (still taking correlation into account) with full posterior
D <- cbind(pilot$theta, pilot$b)[-(1:burnin), ]

adjusted_full_post <- mcmc_mh_cov(
```

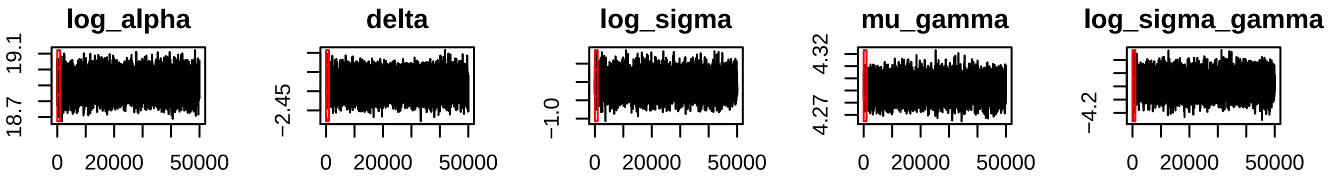

Figure 12: MCMC sampling using full Bayesian posterior

```
  50000, 1000, drop(tail(pilot$theta, 1)), drop(tail(pilot$b, 1)), cov(D), 0.25,
  y = fatigue$N, X = fatigue$s, Z = fatigue$ro
)
```

```
## Total acceptance:        27.174%
## Burned-in acceptance:    27.114%
```

```
# 95% credible intervals
vapply(colnames(adjusted_full_post$theta), function(nm) {
  y <- adjusted$theta[-(1:burnin), nm]
  quantile(y, c(0.025, 0.975))
}, FUN.VALUE = c(0, 0))
```

```
##        log_alpha     delta  log_sigma mu_gamma log_sigma_gamma
## 2.5%    18.77929 -2.404552 -0.8707027 4.275086       -3.953575
## 97.5%   18.95540 -2.332513 -0.7238820 4.297904       -3.511619
```

As expected, these results are very similar to the results in Section 2.4. In both cases, we expect to obtain the same acceptance probabilities, since the $\log(f(\mathbf{n}))$ terms used in the full log posterior in Section 2.5 cancel out: we get an addition and a subtraction in the acceptance probability calculation.

$$\begin{aligned}\log(\text{posterior}_{\text{prop}}) - \log(\text{posterior}_{\text{old}}) &= \log(f(n|\gamma_{\text{prop}})) + \log(\gamma_{\text{prop}}) + \log(f(n)) - \log(f(n|\gamma_{\text{old}})) - \log(\gamma_{\text{old}}) - \log(f(n))) \\ &= \log(f(n|\gamma_{\text{prop}})) + \log(\gamma_{\text{prop}}) - \log(f(n|\gamma_{\text{old}})) - \log(\gamma_{\text{old}})\end{aligned}$$

This final line is what is used in Section 2.4. In reality, when implemented the two calculations are not exactly the same due to floating point differences.