ASI Coursework Report

02340, 02341, 02344

Carcinogenesis study on rats

We analyse the data of a drug trial on 50 litters of female rats. There were three rats per litter, one that recieved a drug treatment and two recieved a control. Let T_i be the follow up time to tumor appearance in rat i, which we assume are independent and identically distributed (iid) Weibull random variables.

We use `optim` with the BFGS method to compute the maximum likelihood estimate of the parameter $\theta^\top = (\beta_0, \beta_1, \log(\sigma))$ as well as the asymptotic standard error of each parameter estimate.

```
nll_weibull <- function(theta,rx,times,status){#negative log-likelihood
  beta0 <- theta[1]; beta1 <- theta[2]; sigma <- exp(theta[3]); eta <- beta0 + beta1*rx
  -sum(dweibull(times[status],shape = 1/sigma, scale = exp(eta[status]), log = TRUE),
       pweibull(times[!status], shape = 1/sigma, scale = exp(eta[!status]),lower.tail = FALSE, log.p = TRUE))
}

theta0 <- c(1,1,1) #initial point
fit_weibull <- optim(par=theta0, fn=nll_weibull, method="BFGS", hessian=TRUE, rx=rx, times=times, status=status)
fit_weibull$convergence #check convergence
```

```
## [1] 0
```

```
mle_params <- fit_weibull$par; mle_params #the optimal values: beta0, beta1, logsigma
```

```
## [1]  4.9831375 -0.2385128 -1.3326129
```

```
-fit_weibull$value #loglikelihood value at the mle
```

```
## [1] -242.2768
```

We compute a 95% asymptotic confidence interval for the treatment effect β_1. The asymptotic distribution of the MLE, $\hat{\theta}$, is $\hat{\theta} \sim N(\theta_t, \mathcal{I}^-1)$, where θ_t is the vector of the true values and $\mathcal{I}$ is the Fisher information matrix. We approximate the information matrix by the Hessian $\nabla^2 l(\hat{\theta})$, where l is the log-likelihood. Using this asymptotic distribution, we get the following 95% confidence interval for β_1:

```
se_weib <- diag(solve(fit_weibull$hessian))^.5 #standard deviations
round(c(mle_params[2] - 1.96*se_weib[2], mle_params[2] + 1.96*se_weib[2]), 4) #confidence interval for beta1
```

```
## [1] -0.4131 -0.0639
```

Interpretation: β_1 is supposed to show the effect of the treatment. The mean of the Weibull distribution is proportional to its scale, which is $e^{\beta_0+\beta_1}$ in case of treatment and e^{β_0} if there is no treatment. Thus if β_1 is positive, the mean of the expected follow up time to tumor is bigger in case of treatment. Similarly, a negative β_1 would show a negative effect (shorter expected follow up time to tumor). The confidence interval shows that with probability 95% β_1 has a slightly negative effect. Therefore, at the 95% confidence level, a rat exposed to the treatment will develop a tumor more quickly, on average, than an otherwise-identical rat which received the control.

We now assume that $T_i \sim$ log-logistic(shape $= 1/\sigma$, scale $= \exp(\eta_i)$), and repeat the above analysis under this new model.

```
nll_llogis <- function(theta,rx,times,status){#negative loglikelihood
  beta0 <- theta[1]; beta1 <- theta[2];sigma <- exp(theta[3]);eta <- beta0 + beta1*rx
  -sum(dllogis(times[status], shape = 1/sigma, scale = exp(eta[status]), log = TRUE),
       sum(pllogis(times[!status], shape = 1/sigma, scale = exp(eta[!status]), lower.tail = FALSE, log.p = TRUE)))
}
theta0 <- c(1,1,1) #initial point
fit_llogis <- optim(par=theta0, fn=nll_llogis, method="BFGS", hessian=TRUE, rx=rx, times=times, status=status)
fit_llogis$convergence #check convergence
```

```
## [1] 0
```

```
mle_llogis <- fit_llogis$par; mle_llogis #the optimal values
```

```
## [1]  4.916533 -0.229507 -1.410281
```

```
-fit_llogis$value #loglikelihood value at the mle
```

```
## [1] -243.3839
```

```
se_llogis <- diag(solve(fit_llogis$hessian))^.5 #standard errors
round(c(mle_llogis[2]-1.96*se_llogis[2], mle_llogis[2]+1.96*se_llogis[2]), 4)#confidence interval for beta1
```

```
## [1] -0.4169 -0.0421
```

Interpretation: The confidence interval shows again that with probability 95% β_1 has a slightly negative effect, so this treatment should not be used. Note, however, that the confidence interval under this model is wider than that under the Weibull model, suggesting that our MLE for β_1 is relatively less precise.

We now assume again that T_i follows a Weibull distribution, but we also try to capture dependence among the rats in each litter by way of a litter-specific random effect $b_{litter(i)}$ in η_i.

Given that η_i is a linear in the parameters and random effects, and letting $\boldsymbol{\eta} = [\eta_1, ..., \eta_n]^T$, we have $\boldsymbol{\eta} = \mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{b}$, which is the linear part of our model. Here, $\boldsymbol{\beta} = [\beta_0, \beta_1]^T$, while the matrices $\mathbf{X}$ and $\mathbf{Z}$ are constructed using the following:

```
X <- model.matrix(~ 1 + rx, data=rats)
rats$litter <- as.factor(rats$litter); Z <- model.matrix(~ litter - 1, data=rats)
```

Suppose that we are given $\mathbf{b}$. If the ith observation is censored, then its contribution to $\log[f(\mathbf{y}|\mathbf{b}, \boldsymbol{\theta})]$ is equal to $-(t_i/\exp(\eta_i))^{1/\sigma}$. Otherwise (i.e. if observation i is uncensored), the resulting contribution to $\log[f(\mathbf{y}|\mathbf{b}, \boldsymbol{\theta})]$ is

$$\log\left\{\left[\frac{1/\sigma}{\exp(\eta_i)}\right]\left[\frac{t_i}{\exp(\eta_i)}\right]^{\frac{1}{\sigma}-1}\right\} - \left(\frac{t_i}{\exp(\eta_i)}\right)^{\frac{1}{\sigma}},$$

which follows directly from the pdf of the Weibull distribution. Let $c_i = 1$ if observation i is censored ($c_i = 0$ otherwise). Then, following some algebraic rearrangement, there holds:

$$\log[f(\mathbf{y}|\mathbf{b}, \boldsymbol{\theta})] = \sum_{i=1}^{n}\left\{c_i\left[\left(\frac{1}{\sigma} - 1\right)\Big(\log(t_i) - \eta_i\Big) - \log(\sigma) - \eta_i\right] - \left[\frac{t_i}{\exp(\eta_i)}\right]^{\frac{1}{\sigma}}\right\}$$

Since $\mathbf{b} \sim N(\mathbf{0}, \sigma_b^2\mathbf{I})$, and using the Normal pdf, $\log[f(\mathbf{b}|\boldsymbol{\theta})] = -\sum_{j=1}^{n_b}\{\log(\sigma_b) + \log(2\pi)/2 + b_j^2/2\sigma_b^2\}$. The R function `lfyb` evaluates the joint log-density $\log[f(\mathbf{y}, \mathbf{b}|\boldsymbol{\theta})] = \log[f(\mathbf{y}, \mathbf{b}|\boldsymbol{\theta})] + \log[f(\mathbf{b}|\boldsymbol{\theta})]$:

```
lfyb <- function(theta, y, b, X, Z) {
  beta <- theta[1:2]; sigma <- exp(theta[3]); sigma_b <- exp(theta[4]) #get parameters
  eta <- as.numeric(X %*% beta + Z %*% b); wshape <- 1/sigma; wscale <- exp(eta) #scale and shape
  time <- y$time; censored <- y$status; #get data (exposure times and censorship flag)
  #log density of y given b, theta
  lfy_b <- sum(censored*((wshape - 1)*(log(time) - eta) - theta[3] - eta) - (time/wscale)^wshape)
  lfb <- -sum(theta[4] + log(2*pi)/2 + b^2/(2*sigma_b^2)) #log density of b given theta
  return(lfy_b + lfb) #log joint density of y, b given theta
}
```

Now, let $L_j = \{i \in \{1, ..., n\} : litter(i) = j\}$. Using the chain rule, we may differentiate the above with respect to $b_j, j \in \{1, ..., 50\}$, from which (and following some simplification) we obtain:

$$\frac{\partial \log[f(\mathbf{y}, \mathbf{b}|\boldsymbol{\theta})]}{\partial b_j} = \frac{1}{\sigma}\sum_{i \in L_j}\left[\left(\frac{t_i}{\exp(\eta_i)}\right)^{\frac{1}{\sigma}} - c_i\right] - \frac{b_j}{\sigma_b^2}, \qquad \frac{\partial^2 \log[f(\mathbf{y}, \mathbf{b}|\boldsymbol{\theta})]}{\partial b_j^2} = -\frac{1}{\sigma^2}\sum_{i \in L_j}\left[\left(\frac{t_i}{\exp(\eta_i)}\right)^{\frac{1}{\sigma}}\right] - \frac{1}{\sigma_b^2}.$$

Further, since the elements of $\mathbf{b}$ are mutually independent, it follows that, if $k \neq j$, then $\partial^2 \log[f(\mathbf{y}, \mathbf{b}|\boldsymbol{\theta})]/\partial b_k \partial b_j = 0$, i.e. $\partial \log[f(\mathbf{y}, \mathbf{b}|\boldsymbol{\theta})]/\partial \mathbf{b} \partial \mathbf{b}^T$ is a diagonal matrix. The R functions `gr_lfyb` and `diagH_lfyb` calculate respectively the gradient and main diagonal vector of $\log[f(\mathbf{y}, \mathbf{b}|\boldsymbol{\theta})]$ with respect to $\mathbf{b}$:

```
gr_lfyb <- function(theta, y, b, X, Z) {
  beta <- theta[1:2]; sigma <- exp(theta[3]); sigma_b <- exp(theta[4])
  eta <- as.numeric(X%*%beta + Z%*%b); wshape <- 1/sigma; wscale <- exp(eta)
  time <- y$time; censored <- y$status
  gr_lfy_b <- wshape*(t(Z) %*% ((time/wscale)^wshape - censored))
  return(gr_lfy_b -b/(sigma_b^2)) #gradient
}

diagH_lfyb <- function(theta, y, b, X, Z) {
  beta <- theta[1:2]; sigma <- exp(theta[3]); sigma_b <- exp(theta[4])
  eta <- as.numeric(X %*% beta + Z %*% b); wshape <- 1/sigma; wscale <- exp(eta)
  time <- y$time;
```

```
  diagH_lfy_b <- -(t(Z) %*% ((time/wscale)^wshape))/(sigma^2)
  return(diagH_lfy_b -1/(sigma_b^2)) #diagonal of Hessian
}
```

The function `lal` evaluates $-\log[f(\mathbf{y}|\boldsymbol{\theta})]$ using the Laplace approximation to the integral $\int \log[f(\mathbf{y},\mathbf{b}|\boldsymbol{\theta})]\mathrm{d}\boldsymbol{\theta}$. The functions `gr_lfyb` and `diagH_lfyb` are used when solving the inner optimisation problem in `lal`. This is more accurate than using finite differences to calculate the gradient and Hessian numerically (which is the default behaviour of `optim`). It is also much quicker; for example, calculating $\partial \log[f(\mathbf{y},\mathbf{b}|\boldsymbol{\theta})]/\partial \mathbf{b}$ using a central finite difference approximation requires an additional $2n_b$ calls to `lfyb`, which we are able to avoid by using the exact gradient.

```
lal <- function(theta, y, X, Z) {
  nb <- ncol(Z) #initial value for b is previous bhat, or zero vector if first run of lal
  b0 <- get0(".initb", envir=environment(lal), ifnotfound=rep(0, length=nb))
  #solve inner optimisation problem
  bhat <- optim(par=b0, fn=lfyb, gr=gr_lfyb, method="BFGS", control=list(fnscale=-1),
                theta=theta, y=y, X=X, Z=Z)
  #bhat from this step is b0 (initial condition) for next step
  assign(".initb", bhat$par, envir=environment(lal))
  #calculate log determinant of Hessian
  logdetH <- sum(log(-diagH_lfyb(theta, y, bhat$par, X, Z)))
  return(logdetH/2 - bhat$value - nb*log(2*pi)/2) #Laplace approximation
}
```

Finally, we may use `optim` to solve for the MLE under this model. As an initial guess for $(\beta_0, \beta_1, \log(\sigma))^T$, we use the maximum likelihood estimate for the Weibull model with no random effects. The resulting MLE in the random effects setting is:

```
mle_lal <- optim(par=c(mle_params, -1), fn=lal, method="BFGS",
                 control=list(trace=1, maxit=1000, ndeps=rep(1e-2, length=4)), hessian=TRUE,
                 y=rats, X=X, Z=Z) #MLE for (beta0, beta1, log(sigma), log(sigma_b))'
```

```
## initial  value 244.957238
## iter  10 value 241.225271
## iter  10 value 241.225717
## final  value 241.225271
## converged
```

```
mle_lal$par
```

```
## [1]  5.0091433 -0.2307949 -1.3787137 -1.6512501
```

The lower and upper bounds of a confidence interval around the treatment effect β_1 are given below, and are calculated using the asymptotic normality of the MLE:

```
mle_b1 <- mle_lal$par[2]; sd_mle_b1 <- sqrt(solve(mle_lal$hessian)[2,2]) #standard error
round(mle_b1 + 1.96*sd_mle_b1*c(-1,1), 4) #95% CI
```

```
## [1] -0.4034 -0.0582
```

Once again, we conclude that, with 95% confidence, β_1 has a negative impact on the average time before a rat in the treatment group develops a tumour. The confidence interval under this model is also slightly narrower than in the Weibull model without random effects considered above. This suggests that at least some of the variation in outcomes is litter-specific, and that including the random effects to try to control for this variation has allowed us to (very slightly) improve the precision of our MLE for β_1.

We now move to a Bayesian setting; assume the same model as above, but that $\theta = (\beta_0, \beta_1, \log(\sigma), \log(\sigma_b))^T$ is a random vector. We assume that σ_b follows an exponential prior with rate 5 and is independent of the other parameters, which each follow improper uniform priors. This gives us the following log posterior function:

```
log.post <- function(beta0, beta1, log_sigma,log_sigma_b,b_tumour, b_dead,  tumour, dead) {
  log_pi0_beta0=log(1);  log_pi0_beta1=  log(1);  log_pi0_log_sigma=log(1);
  log_pi0_log_sigma_b= log(5*exp(-5*exp(log_sigma_b))) # log of the priors
    log_pi0_b_tumour=log(dnorm(b_tumour, mean=0, sd=exp(log_sigma_b)));
    log_pio_b_dead=log(dnorm(b_dead, mean=0, sd=exp(log_sigma_b))) # log of the priors
  log_pi0<- log_pi0_beta0+log_pi0_beta1+log_pi0_log_sigma+log_pi0_log_sigma_b+sum(log_pio_b_dead)+
    sum(log_pi0_b_tumour)# we add the priors since we assume they are independent
  log.lik<-sum(log (dweibull(tumour$time, 1/exp(log_sigma), exp(beta0+beta1*tumour$rx+b_tumour))))+
    sum(log (pweibull(dead$time, 1/exp(log_sigma), exp(beta0+beta1*dead$rx+b_dead), lower.tail=FALSE)))# Now the l
  return(log.lik+log_pi0)# now the log posterior = log likelihood +log prior
}
```

We use a normal centered on the current values as the proposal for $\boldsymbol{\theta}$, and each element of b I will also use a symmetric multivariate normal distribution centered on the current values for the values of b. We will need to tune the proposal standard deviations to get appropriate acceptance rates, aiming for about 25%.

The initial value is important, since we would like to start in a region of the parameter space with high density. Otherwise (i.e. if the posterior density is extremely low for the initial value), it will take us a long time (a large number of generated values) to reach the area of high posterior density and therefore a long time to start sampling from the stationary distribution of the Markov chain (the posterior). Thus we choose as our initial conditions the maximum likelihood estimators calculated earlier in the random effects model.

We choose a burn in period of 10,000 but this can always be changed later.

```
split_rats <- split(rats, rats$status)
tumour=as.data.frame(split_rats[[2]]); dead=as.data.frame(split_rats[[1]])
nsteps=100000 ; burn.in=10000
MH<-function(beta0_0,beta1_0, log_sigma_0, log_sigma_b0, b_tumour0, b_dead0, sigma_beta0, sigma_beta1,
             sigma_log_sigma, sigma_log_sigma_b,   sigma_b, nsteps){
  accept <- rep(0,3) # set up locations to store values at each step
  beta0 <- rep(0,nsteps) ;  beta1=rep(0,nsteps);  log_sigma=rep(0,nsteps);  log_sigma_b=rep(0,nsteps)
  b_tumour=matrix(0,nsteps,length(tumour$rx)) ;  b_dead=matrix(0,nsteps,length(dead$rx))
  beta0[1] <- beta0_0 ;  beta1[1]=beta1_0;  log_sigma[1]=log_sigma_0;  log_sigma_b[1]=log_sigma_b0;
  b_tumour[1,]=b_tumour0;  b_dead[1,]=b_dead0   #Set intial values
  lp0 <- log.post(beta0_0,beta1_0, log_sigma_0, log_sigma_b0,b_tumour0, b_dead0,tumour, dead) #initial log post.
for(i in 2:nsteps){ #MH loop
current_beta0=beta0[i-1] ;current_beta1=beta1[i-1];current_log_sigma=log_sigma[i-1];  # set current values
current_log_sigma_b=log_sigma_b[i-1];current_b_tumour=b_tumour[i-1,]; current_b_dead=b_dead[i-1,]
proposed_log_sigma_b=current_log_sigma_b+rnorm(1,0,sigma_log_sigma_b)#update sigma_b
lp1 <- log.post(current_beta0,current_beta1, current_log_sigma, proposed_log_sigma_b,current_b_tumour,
                current_b_dead,tumour, dead) # find log posterior of new values
acc <- exp(min(0,lp1-lp0))
if (runif(1)>=acc| !is.finite(acc)){#reject
  b_tumour[i,] <- current_b_tumour ;  b_dead[i,]=current_b_dead;  beta0[i] <- current_beta0;
  beta1[i]=current_beta1;  log_sigma[i]=current_log_sigma;  log_sigma_b[i]=current_log_sigma_b
  lp1<- lp0 ## Return to the 'old' log posterior
}else {#accept
  accept[1]=accept[1]+1 # keep track of number of acceptances
  log_sigma_b[i]=proposed_log_sigma_b #store found values
  lp0 <- lp1 ## uldate old log posterior to the new one
    proposed_b_tumour=current_b_tumour + rnorm( length(tumour$rx), mean=0, sd=sigma_b)#update b
    proposed_b_dead=current_b_dead+ rnorm( length(dead$rx), mean=0, sd=sigma_b)#update b
    lp1 <- log.post(current_beta0,current_beta1, current_log_sigma, proposed_log_sigma_b,proposed_b_tumour,
                    proposed_b_dead,tumour, dead) # update log posterior
  acc <- exp(min(0,lp1-lp0))
  if (runif(1)>=acc | !is.finite(acc)){#reject
  b_tumour[i,] <- current_b_tumour ;  b_dead[i,]=current_b_dead;  beta0[i] <- current_beta0;
  beta1[i]=current_beta1;  log_sigma[i]=current_log_sigma     #store values
  lp1<- lp0 ## Return to previous log posterior
}else {#accept
          accept[2]=accept[2]+1 # keep track to calculate acceptance rates
          b_tumour[i,] <- proposed_b_tumour;  b_dead[i,]=proposed_b_dead;  #store values
          lp0 <- lp1 ## update log posterior
  proposed_beta0=current_beta0+rnorm(1,0,sigma_beta0); proposed_beta1=current_beta1+rnorm(1,0,sigma_beta1)
  proposed_log_sigma=current_log_sigma+rnorm(1,0,sigma_log_sigma)#update remaining paramaters
  lp1=log.post(proposed_beta0,proposed_beta1, proposed_log_sigma, proposed_log_sigma_b,
               proposed_b_tumour, proposed_b_dead,tumour, dead) # calculate new log posterior
  acc <- exp(min(0,lp1-lp0))
  if (runif(1)>=acc| !is.finite(acc)){#reject
  beta0[i] <- current_beta0 ;  beta1[i]=current_beta1;  log_sigma[i]=current_log_sigma #store values
  lp1<- lp0 ## Return to previous log posterior
}else {#accept
    accept[3]=accept[3]+1 # keep track to calculate acceptance rates
  beta0[i] <- proposed_beta0;  beta1[i]=proposed_beta1;  log_sigma[i]=proposed_log_sigma#store values
  lp0=lp1 # update log posterior
}}}}
  list(beta0=beta0, beta1=beta1, log_sigma_b=log_sigma_b, log_sigma=log_sigma,ar_outer=accept[1]/nsteps,
       ar_middle=accept[2]/accept[1], ar_inner=accept[3]/accept[2])
```

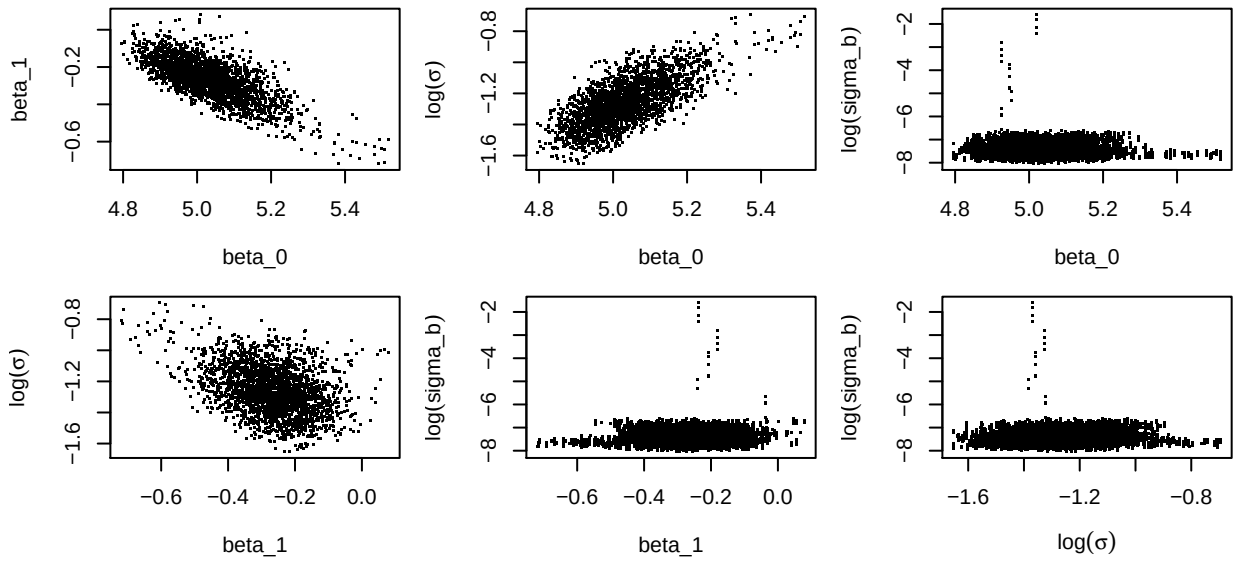
```
}
mh=MH(5.0188886, -0.2376741, -1.3687252,-1.5976068, rep(0, length(tumour$rx)),rep(0, length(dead$rx)),
      0.1,0.1,0.1,0.25,0.0001, nsteps = nsteps)
```

For tuning the proposal distribution we use run lengths of about 10000. 100000 would be better for the final run. The aim is for an acceptance rate of approximately 25%.

```
c(mh$ar_outer,mh$ar_middle,mh$ar_inner)
```

```
## [1] 0.2763800 0.2861278 0.2979262
```

We check for correlation between the parameters by plotting the sampled values, with one plot for each possible pair.



We note the elliptical shaped graphs especially for the plots of $\beta_0, \beta_1, \log(\sigma), \log(\sigma_b)$. This suggests that the posterior density for θ is highly non independent, and independent jumps for each component will give slow mixing. We consider using a shrunken version of the co-variance, found using the results above, as the basis for proposing multivariate normal jumps in a random walk i.e. $\theta_i \sim N(\theta_{i-1}, \lambda * \Sigma)$, where $\lambda > 0$ and we can tune. To find the co-variance matrix, Σ:

```
library(mvtnorm)
mu=c(mean(mh$beta0[-(burn.in)]),mean(mh$beta1[-(burn.in)]), mean(mh$log_sigma[-(burn.in)]) )
s11=cov(mh$beta0[-(burn.in)], mh$beta0[-(burn.in)]);s12=cov(mh$beta0[-(burn.in)], mh$beta1[-(burn.in)])
s13=cov(mh$beta0[-(burn.in)], mh$log_sigma[-(burn.in)]);s22=cov(mh$beta1[-(burn.in)], mh$beta1[-(burn.in)])
s23=cov(mh$beta1[-(burn.in)], mh$log_sigma[-(burn.in)]);s33=cov(mh$log_sigma, mh$log_sigma)
covariance= matrix(c(s11,s12,s13,s12,s22,s23,s13,s23,s33), nrow=3, byrow=TRUE)
```

With this new proposal distribution for $(\beta_0, \beta_1, \log(\sigma)$ we get a new MH sampler identical to the previous one, except that when we propose new values in the inner most MH chain we use this excerpt of code:

```
proposed= rmvnorm(1, mean=c(current_beta0, current_beta1, current_log_sigma),
                  lambda*covariance) #update remaining paramaters
proposed_beta0=proposed[1];  proposed_beta1=proposed[2];  proposed_log_sigma=proposed[3]
```

Using the new MH algorithm and tuning we get

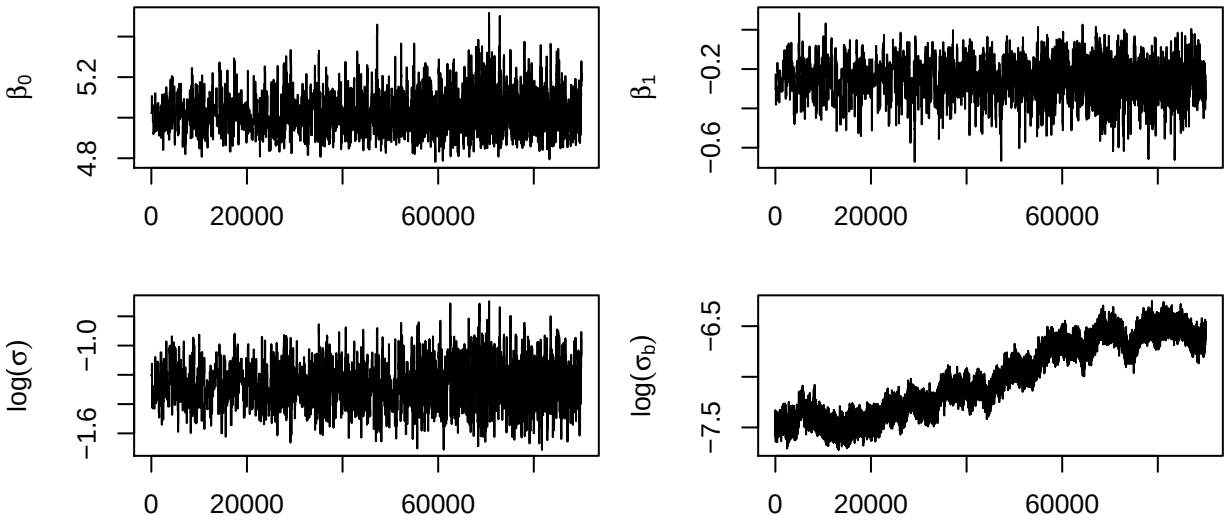
```
mh=MH2(5.0188886, -0.2376741, -1.3687252,-1.5976068, b_tumour0 =  rep(0, length(tumour$rx)),
       b_dead0= rep(0, length(dead$rx)),lambda=2.5,sigma_log_sigma_b = 0.25, sigma_b =0.0001,
       nsteps = nsteps)
mh$ar_outer;mh$ar_middle;mh$ar_inner
```

```
## [1] 0.27576
```

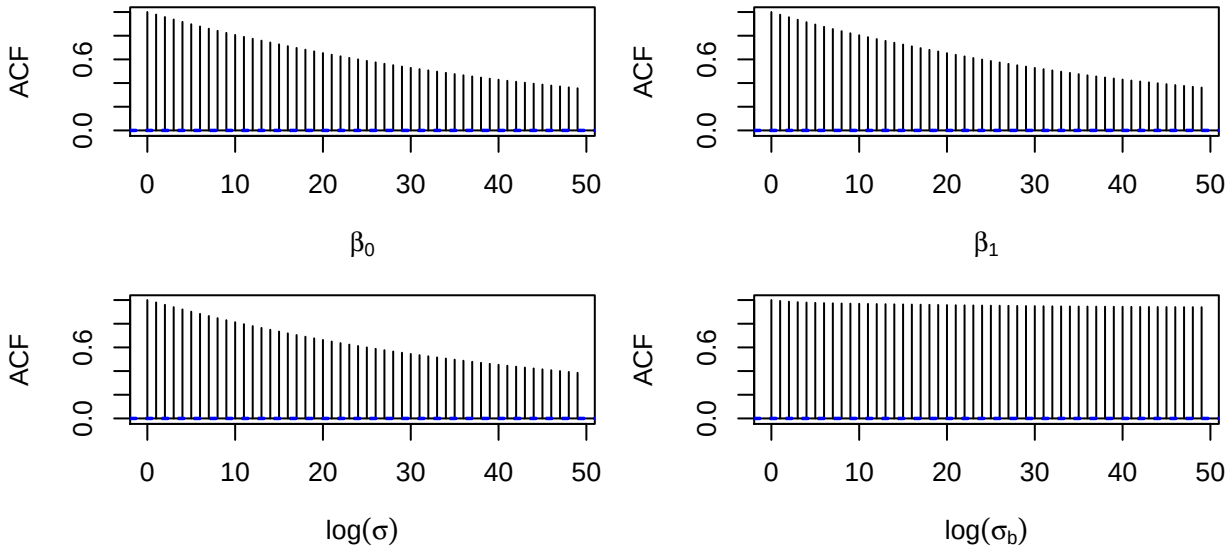
```
## [1] 0.4578619
```

```
## [1] 0.248614
```

We now check that the Markov chain is behaving as we expect. Firstly, the trace plots show some good mixing within the parameters.



The auto-correlation function plots are also decreasing suggesting that the Markov chain is converging.



We can also get some idea of the effective sample size for each of our parameters:

```
n.eff <- c(0,0,0,0)
autocor <- acf(mh$beta0[-(burn.in)],plot=FALSE); t.eff <- 2*sum(autocor[[1]]) - 1; n.eff[1] <- nsteps/t.eff
autocor <- acf(mh$beta1[-(burn.in)],plot=FALSE); t.eff <- 2*sum(autocor[[1]]) - 1; n.eff[2] <- nsteps/t.eff
autocor <- acf(mh$log_sigma[-(burn.in)],plot=FALSE); t.eff <- 2*sum(autocor[[1]]) - 1;n.eff[3] <- nsteps/t.eff
autocor <- acf(mh$log_sigma_b[-(burn.in)],plot=FALSE);t.eff <- 2*sum(autocor[[1]]) - 1 ;n.eff[4] <- nsteps/t.eff
n.eff
```

```
## [1] 1632.759 1633.671 1596.081 1056.299
```

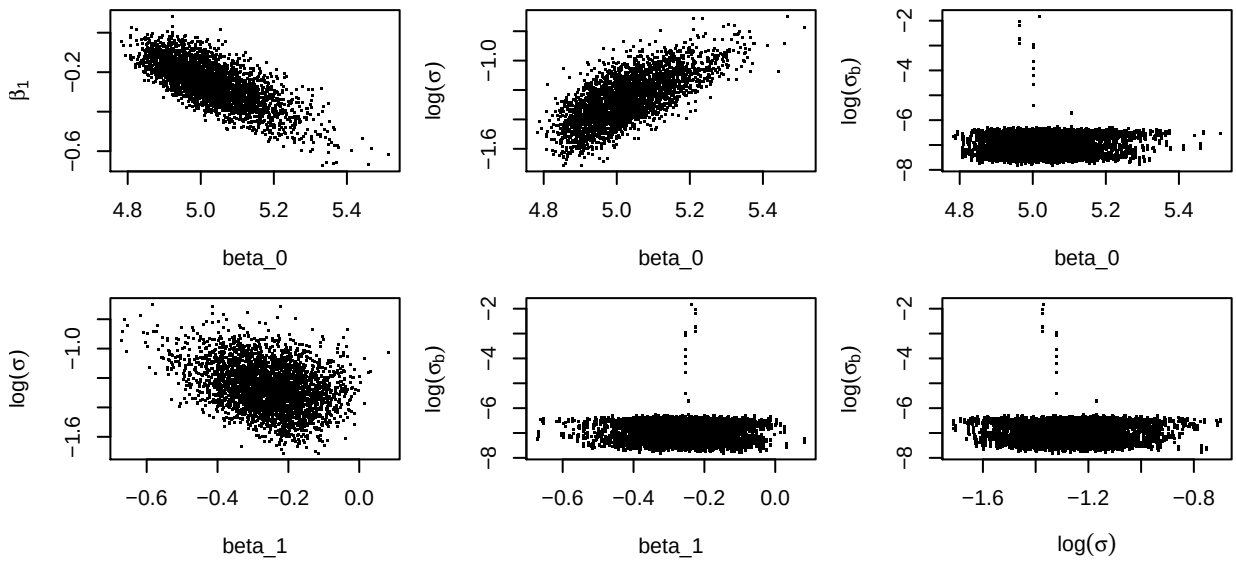
We will also obtain an independent sub-sample of our chain and then split it into two sub-samples. Using the Kolmogorov-Smirnov test we hope to see that they come from the same distribution and hence that the Markov chain has converged. The null hypothesis is that each sample comes from the same distribution. If we reject this null, then that implies that (at least) one of the sub-samples is not being sampled from the posterior distribution, and hence that the Markov chain has not converged.

```
k1=ks.test(mh$beta0[-1000:-500], mh$beta0[-500]);
k2=ks.test(mh$beta1[-1000:-500], mh$beta1[-500]);
k3=ks.test(mh$log_sigma_b[-1000:-500], mh$log_sigma_b[-500]);
k4=ks.test(mh$log_sigma[-1000:-500], mh$log_sigma[-500]);
c(k1$p.value,k2$p.value,k3$p.value,k4$p.value)
```

```
## [1] 0.9995397 0.9952896 0.6336954 0.9945130
```

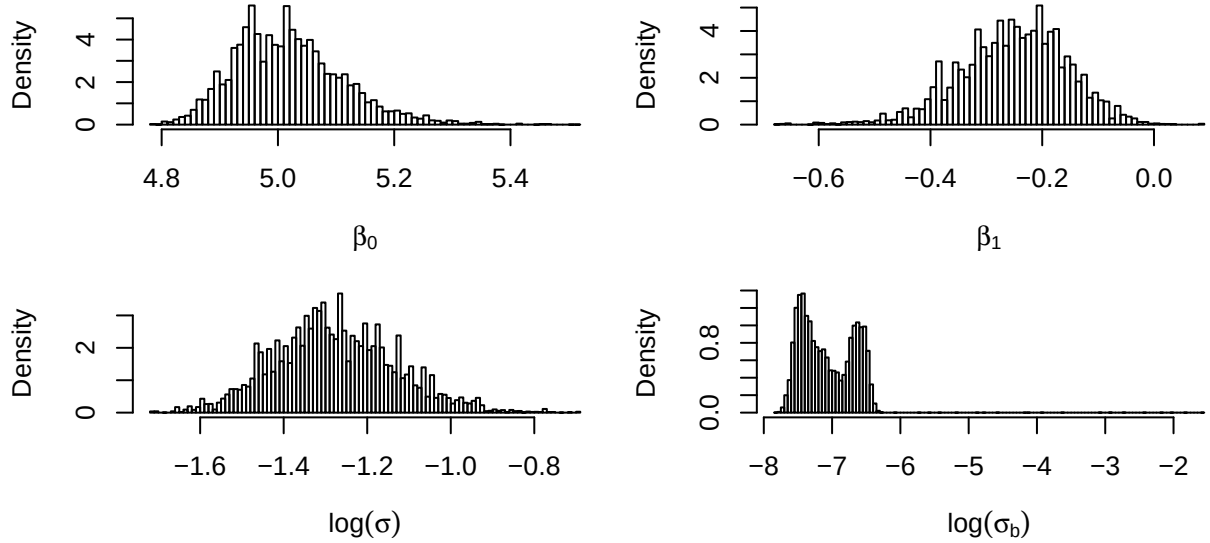
We see that in all cases the p-values are large, and thus there is not significant evidence that the two sub-samples are from different distributions. Hence we conclude that the Markov chain has converged.

Again, we investigate the correlation between the parameters by plotting a random sample of the iterations retained after discarding the burn in period .



It looks like we need a more sophisticated proposal to deal with the correlation of $\log(\sigma_b)$ and the other 3 parameters.

We also investigate the shape of the marginal posterior densities of the parameters by plotting histograms of the retained values



The long non symmetric tails on some of the marginal posterior densities suggesting such as on the $\log(\sigma_b)$ graph again suggest a more sophisticated proposal would be useful. We compute a 95% posterior probability interval for the intervention effect β_1.

```
quantile(mh$beta1[-(burn.in)], c(.025, 0.975))#95% condfidence interval for beta_1
```

```
##        2.5%       97.5%
## -0.45143851 -0.07764451
```

We conclude that 0 is not contained in the 95% posterior probability interval for the intervention effect, suggesting that the intervention has a statistically significant effect. As per the analysis in part 2, the tails for the credible interval for β_1 being less than zero suggest that the treatment is having a negative effect. This corresponds well with the previous results for the non-Bayesian models.

Fatigue of materials

We now consider materials used in mechanical and structural engineering, which are required to withstand certain cyclic stresses. Small flat plates (coupons) of a nickel base superalloy have been subjected to tests for their strength, with each coupon i being placed under cyclic loads of a fixed level of stress, S_i. The number of cycles to failure, N_i, is then recorded for each coupon. There are also some cases where failure was not recorded even after N_i cycles; these are referred to as run-outs, and are treated in the same way as censored observations in an analysis of survival data (such as the carcinogenesis data for rat litters considered above).

Assume throughout that $N_i \sim \text{Weibull}\big(\text{shape} = 1/\sigma, \text{scale} = \alpha(s_i - \gamma)^\delta\big)$, where $s_i > gamma$, and γ denotes some minimum stress threshold below which the material will never fail. We use `optim` to find the maximum likelihood estimate of $\boldsymbol{\theta} = (\log(\alpha), \delta, \log(\sigma))^\top$ for an arbitary value of γ. We compute asymptotic 95% confidence intervals for each of the unknown parameters and investigate the sensitivity of the results to the chosen value of γ.

```
#negative log likelihood
nll_weibull2 <- function(theta,gam,s,N,ro) {
  alpha <- exp(theta[1]); delta <- theta[2]; sigma <- exp(theta[3])
  -sum(dweibull(N[!ro], shape = 1/sigma, scale = alpha*(s[!ro]-gam)^delta, log = TRUE),
       sum(pweibull(N[ro], shape = 1/sigma, scale = alpha*(s[ro]-gam)^delta, lower.tail=FALSE, log.p=TRUE)))
}
#MLE with optim - Set initial point and choose a gamma. min(s)=80.3, so we can try gam=80.
theta0 <- c(1,1,1); gam <- 80
fit_weibull2 <- optim(par=theta0, fn=nll_weibull2, method="BFGS", hessian=TRUE, gam=gam, s=s, N=N, ro=ro)
fit_weibull2$convergence #check convergence
```

```
## [1] 0
```

```
mle_params2 <- fit_weibull2$par; mle_params #the optimal values
```

```
## [1]  4.9831375 -0.2385128 -1.3326129
```

```
-fit_weibull2$value #loglikelihood value at the mle
```

```
## [1] -255.6109
```

```
#standard errors and conf interval: similarly as in the "rats" problem
se_weibull2 <- diag(solve(fit_weibull2$hessian))^.5 #standard deviations
ci <- matrix(nrow=3, ncol=2);
rownames(ci) <- c("log alpha", "delta", "log sigma"); colnames(ci) <- c("lower", "upper")
for(i in 1:3) { #confidence interval for each parameter in theta
  ci[i,] <- round(mle_params2[i] + 1.96*se_weibull2[i]*c(-1,1), 3)
}; ci
```

```
##            lower  upper
## log alpha 12.741 14.404
## delta     -1.342 -0.749
## log sigma -0.867 -0.155
```

The function `gamma_sensitivity` (code is omitted here for brevity) calls `optim` for a given initial point $\boldsymbol{\theta}_0$ and for several values of γ. It then plots the optimal values of the different parameters as a function of γ, and returns a data frame containing as columns all γ values, the optimal log-likelihood values, the ML estimates, and the mean of the scale parameter in case of each γ. So the mean scale parameter is given by `mean(alpha(s - gam_list[ind])^ delta)` (the mean is taken over the s values).

```
theta0 = c(1,1,1); gam_list <- seq(1,80,0.1)
sens_data <- gamma_sensitivity(theta0, gam_list)
#the log-likelihood has maximum at:
sens_data$gamma[which(sens_data$log_like %in% max(sens_data$log_like))]
```

```
## [1] 66.5
```

```
#optimal values of the other parameters at the same gamma:
sens_data$logalpha[which(sens_data$log_like %in% max(sens_data$log_like))]
```

```
## [1] 18.24459
```

```
sens_data$delta[which(sens_data$log_like %in% max(sens_data$log_like))]
```

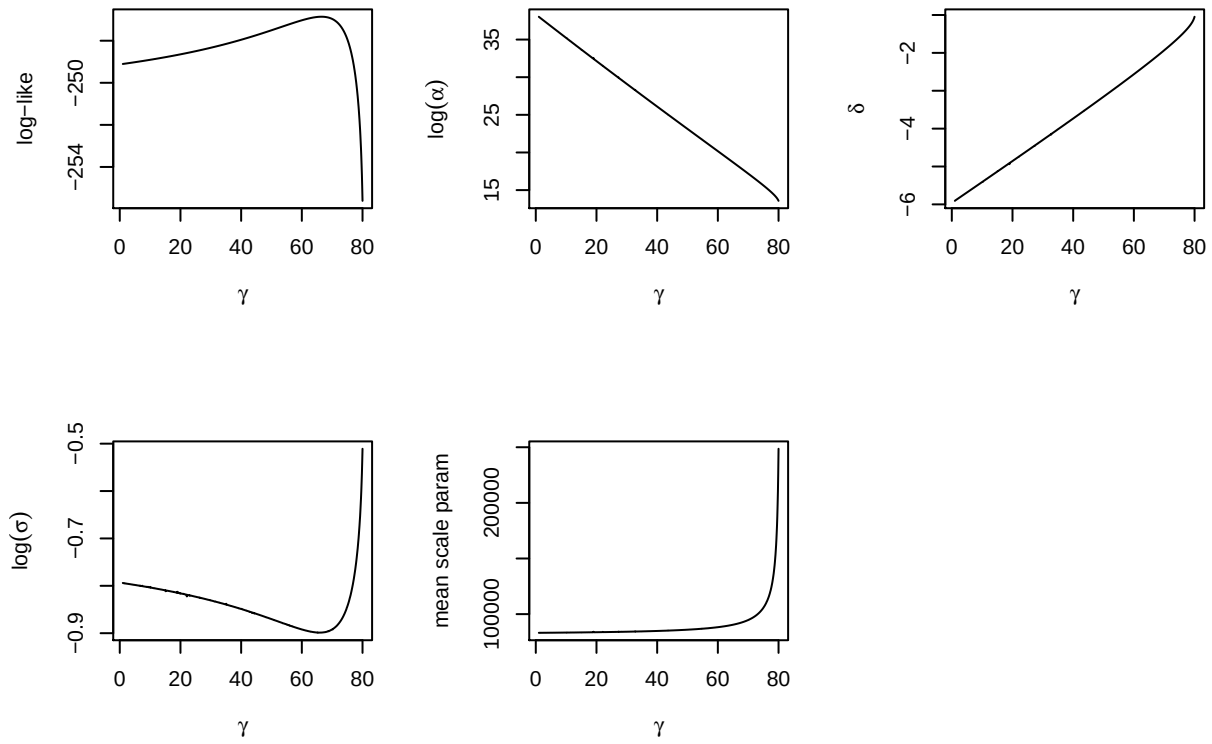
```
## [1] -2.160232
```

```
sens_data$logsigma[which(sens_data$log_like %in%  max(sens_data$log_like))]
```

```
## [1] -0.8986544
```

```
#logsigma has minimum at:
sens_data$gamma[which(sens_data$logsigma %in% min(sens_data$logsigma))]
```

```
## [1] 65.2
```



The log-likelihood has maximum at $\gamma = 66.5$ and the negative log-likelihood has almost the same shape as $\log(\sigma)$. This is due to the fact that the scale parameter is very smooth except when γ is close to 80, so the shape parameter determines the behavior of the likelihood function.

Now consider the behaviour of the shape parameter for the Weibull distribution. $\log(\sigma)$ has a minimum (and the shape parameter a maximum) at $\gamma = 65.2$. As long as the (mean of the) scale parameter changes moderately, the decrease can be explained as follows. When γ is small, the minimum stress level is much higher than the fatigue limit. Therefore early failures are more likely even at lower levels of stress. This would mean a greater positive skewness in the Weibull distribution, i.e. a smaller shape parameter (bigger $\log(\sigma)$). However, if γ is large, then the fatigue limit is close to the minimum stress level. Thus it is more likely that it takes some time for failure to occur at stress levels close to the limit. This gives smaller positive skewness, and a larger value for the shape parameter (smaller $\log(\sigma)$). As soon as the mean scale parameter starts to increase more quickly (around $\gamma > 66$) this explanation fails, and positive skewness occurs again.

Similarly, we now consider the scale parameter; $\log(\alpha)$ is in the range $(13, 40)$ and decreases linearly as a function of the chosen γ (meaning that the α values are large and decrease exponentially with γ). δ is in the range $(-6, -1)$ and increases linearly as a function of the chosen γ. The fact that α and δ react in opposite ways to changes in the value of γ keeps the scale parameter on the same order of magnitude for different values of γ (linearity fails when γ is close to 80). However, as γ increases, the scale parameter increases (slowly) as well. This should be intuitive, because we expect more time to elapse until failure, if the minimum stress level is closer to the fatigue limit (and the scale parameter is proportional to the expectation of the Weibull distribution).

We now estimate the full vector of unknown parameters - that is, we include γ in our unknown parameter vector θ. By definition of γ (as the fatigue limit), we impose the constraint $0 < \gamma < \min(s[!ro]) = \min(s)$. Let $\theta_4 = \text{logit}\left(\frac{\gamma}{\min(s)}\right)$, i.e. we map $\gamma/\min(s)$ from (0,1) to $\mathbb{R}$, to make the optimisation work.

```
#new negative log-likelihood with new theta
nll_weibull2_gam <- function(theta,s,N,ro){
  alpha <- exp(theta[1]);delta <- theta[2];sigma <- exp(theta[3])
  gam <- min(s)*1/(exp(-theta[4])+1) #inverse logit
  #other possibilities give similar results, e.g.:
  #gam <- min(s)*exp(-exp(theta[4]));gam <- min(s)*pnorm(theta[4])
  -sum(dweibull(N[!ro], shape = 1/sigma, scale = alpha*(s[!ro]-gam)^delta, log = TRUE),
       pweibull(N[ro], shape = 1/sigma, scale = alpha*(s[ro]-gam)^delta, lower.tail = FALSE, log.p = TRUE))
}
theta0 <- c(18.24,-2.16,-0.9,log(66.5/min(s))-log(1-66.5/min(s))) #use the values we got from gamma_sensitivity
fit_weibull2_gam <- optim(par=theta0, fn=nll_weibull2_gam, method="BFGS", hessian=TRUE, s=s, N=N, ro=ro)
fit_weibull2_gam$convergence #check convergence
```

```
## [1] 0
```

```
mle_gam <- fit_weibull2_gam$par; mle_gam #opt parameters
```

```
## [1] 18.2420087 -2.1596732 -0.8986556  1.5732532
```

```
-fit_weibull2_gam$value #loglikelihood value at the mle
```

```
## [1] -246.8603
```

```
min(s)*1/(exp(-mle_gam[4])+1) #opt gamma
```

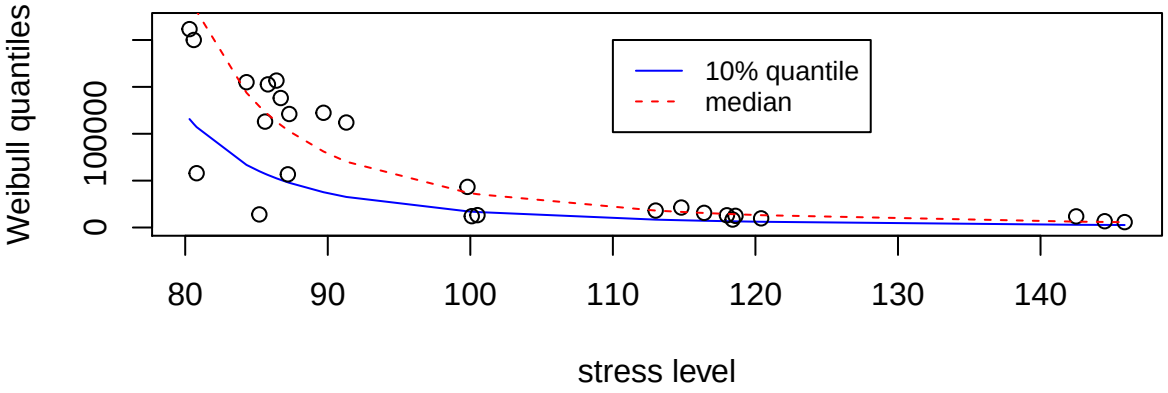
```
## [1] 66.50822
```

The 95% confidence intervals for $\log(\alpha)$, δ, $\log(\sigma)$ and $\text{logit}(\gamma/\min(s))$ are given below.

```
##                       lower   upper
## log alpha          12.9363 23.5478
## delta              -3.3093 -1.0100
## log sigma          -1.2431 -0.5542
## transformed gamma   0.0515  3.0950
```

We now investigate the lower 10% quantile of N which in this case is given by $N_{0.1} = \alpha\,(s_i - \gamma)^{\delta}\, z_{0.1}^{\sigma}$. We estimate and plot the lower 10% quantile curve of N and plot it together with the data and the median of an exponential distribution with unit rate.

```
#mle results:
alpha <- exp(mle_gam[1]); delta <- mle_gam[2]
sigma <- exp(mle_gam[3]);gam <- min(s)*1/(exp(-mle_gam[4])+1)
#plot 10% and 50 % quantiles as a function of s
weibull2_quantiles10 <- alpha*(s-gam)^delta*
  qexp(0.1, rate = 1, lower.tail = TRUE, log.p = FALSE)^sigma
weibull2_quantiles50 <- alpha*(s-gam)^delta*
  qexp(0.5, rate = 1, lower.tail = TRUE, log.p = FALSE)^sigma
```



Consider now the following Bayesian random-effects model that allows the fatigue limit to be different for each coupon. This is achieved by modelling the fatigue limit as an unobserved random variable Γ_i, with its own Weibull distribution. We must now also make the distribution of N_i conditional on $\Gamma_i = \gamma_i$, but this remains the Weibull distribution considered above. We also assume that $\boldsymbol{\theta}$ is now a random vector; σ_b follows an exponential prior with rate 5, independently of the other parameters which all have improper uniform priors.

We wish to use a Metropolis-Hastings algorithm to sample from the posterior distribution of θ and the random effects b.

As in the previous section, we constrct a Metropolis-Hastings algorithm to sample from the posterior distribution (of $\boldsymbol{\theta}|\mathbf{y}$) and the random effects $\mathbf{b}$. The below computes a log posterior function for the model given parameters: $\log(\alpha), \delta, \log(\sigma), \mu_\gamma, \log(\sigma_\gamma)$ and γ

```
log.post <- function(log_alpha, delta, log_sigma,mu_gamma,log_sigma_gamma,gamma_broke, gamma_runoff,
                     broke, runoff) {
  log_pi0_log_alpha=log(1);log_pi0_delta=  log(1);log_pi0_log_sigma=log(1);log_pi0_mu_gamma= log(1) #log prior
  log_pi0_log_sigma_gamma=log(5*exp(-5*exp(log_sigma_gamma)))
  log_pi0_gamma_broke=log(dweibull(gamma_broke, 1/exp(log_sigma_gamma), exp(mu_gamma)))#log prior
  log_pi0_gamma_runoff=log(dweibull(gamma_runoff, 1/exp(log_sigma_gamma), exp(mu_gamma)))#log prior
  log_pi0<- log_pi0_log_alpha+log_pi0_delta+log_pi0_log_sigma+log_pi0_mu_gamma+log_pi0_log_sigma_gamma +
    sum(log_pi0_gamma_runoff)+sum(log_pi0_gamma_broke)# we add the priors since we assume they are independent
  log.lik=sum(log(pweibull(broke$N, 1/exp(log_sigma), exp(log_alpha)*(broke$s-gamma_broke)^delta, lower.tail=FALSE)
    sum(log(dweibull(runoff$N, 1/exp(log_sigma), exp(log_alpha)*(runoff$s-gamma_runoff)^delta)))# Now the log like
  return(log.lik+log_pi0) # now the log posterior = log likelihood +log prior
}
```

We now move onto the MH algorithm. As in the previous section, we use a Normal proposal distribution centred on the previous value in the chain, and tune the sampler with standard deviations for each component. For the random effects $\mathbf{b}$, we use a uniform proposal centred on the previous value, and tune the sampler using the range of this distribution. The burn-in period is 10000 samples. We tune

the parameters to try and get acceptance rates of around 25% although as noted earlier, the system is quite sensitive to values of γ making the tuning very difficult to do.

In the interests of brevity the full Metropolis-Hastings sampler has been omitted from this report, as it is quite similar to the one used in the previous section. The full code is available in the R-Markdown source.

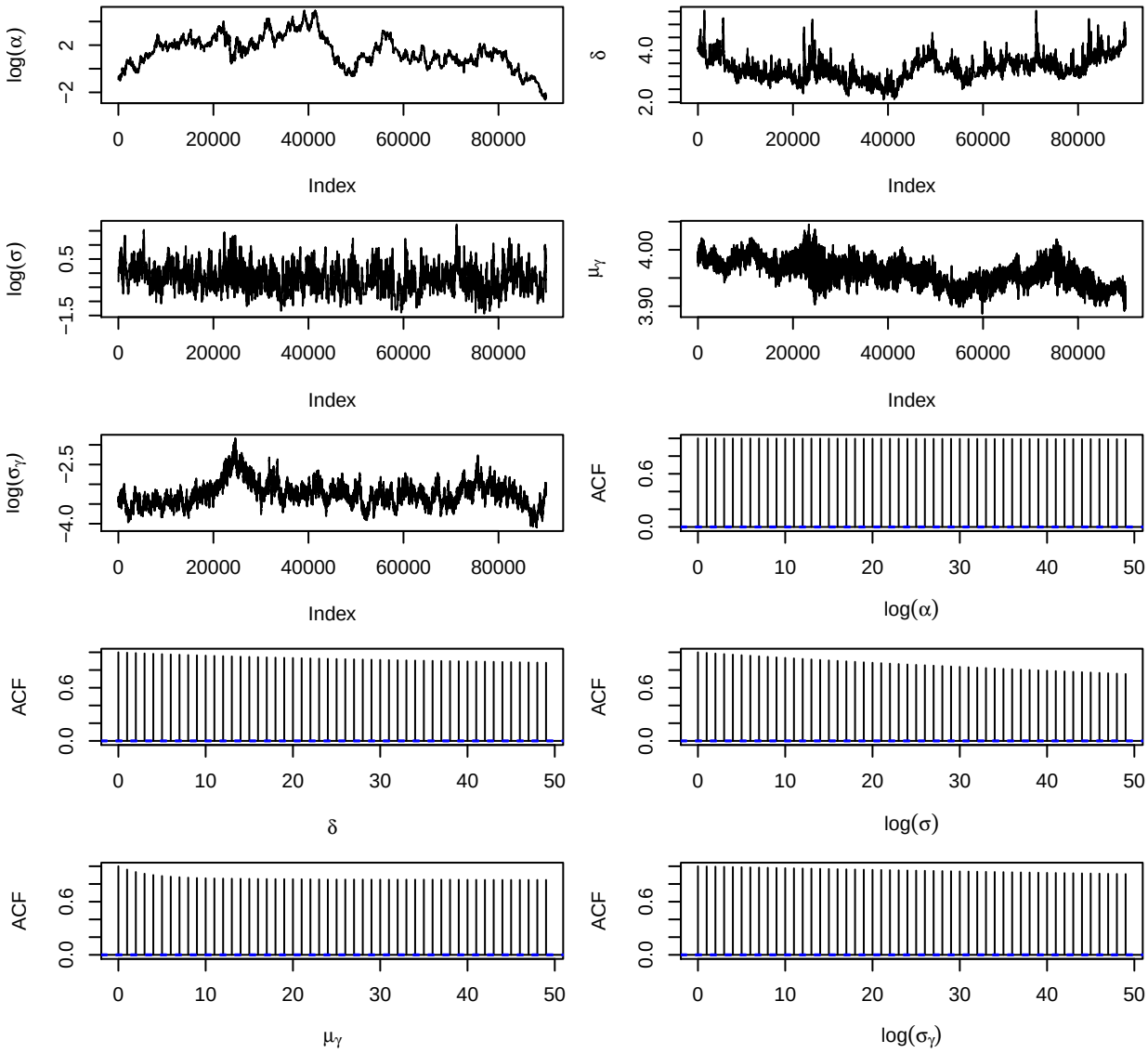
```
mh=MH(0,1,1,4,-3,rep(50, length(broke$N)),rep(50, length(runoff$N)),0.1,0.3,0.3,0.05,0.05,0.4,nsteps = nsteps)
mh$ar_outer;  mh$ar_middle;  mh$ar_inner
```

```
## [1] 0.18624
```

```
## [1] 0.7811963
```

```
## [1] 0.3740463
```

We plot the trace graphs of our retained results (minus the burn-in period) and the autocorrelation functions:

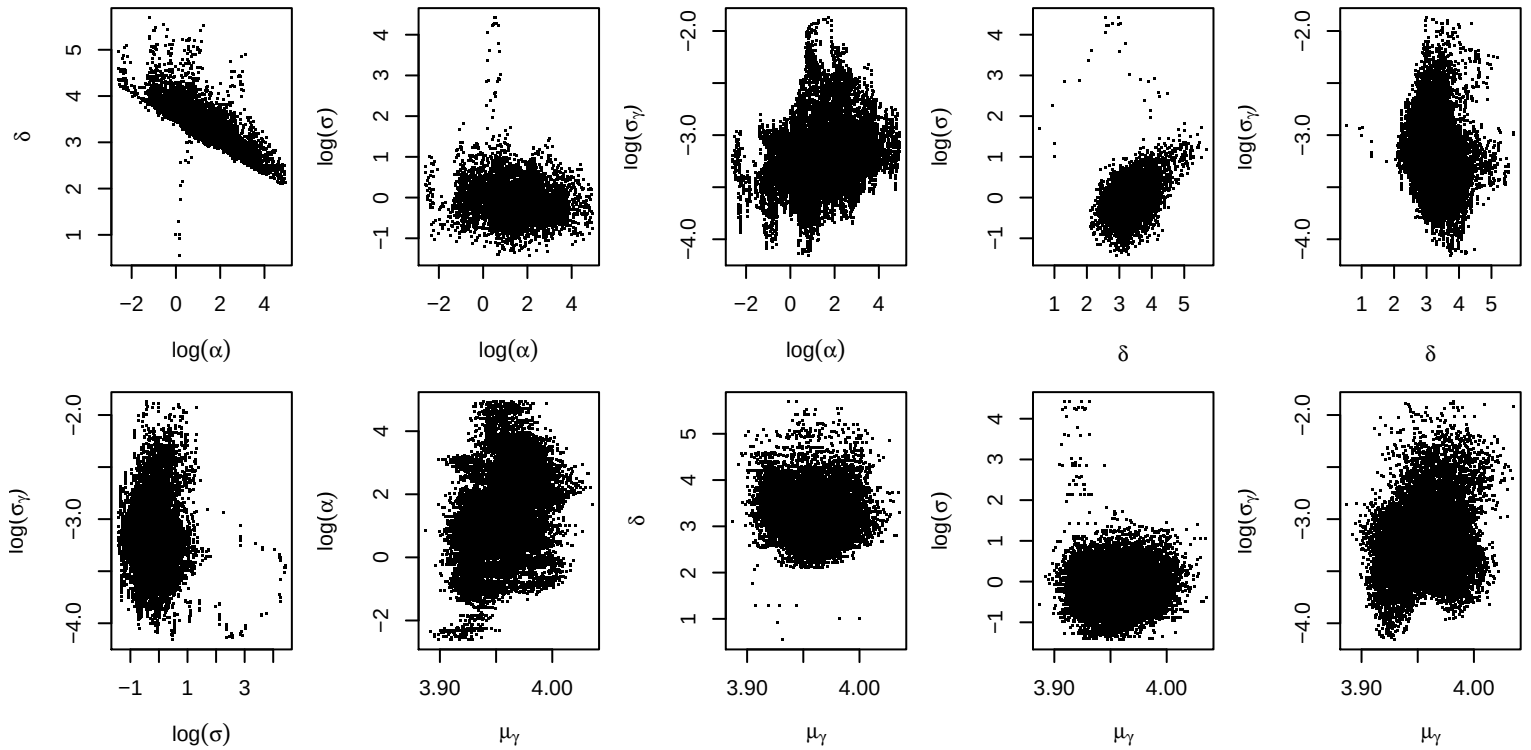


We see that we get quite small values for effective sample size and that the sample size for $\log(\sigma_\gamma)$ is extremely small.

```
n.eff <- c(0,0,0,0,0)
autocor <- acf(mh$log_alpha[-(burn.in)],plot=FALSE);t.eff <- 2*sum(autocor[[1]]) - 1;n.eff[1] <- nsteps/t.eff
autocor <- acf(mh$delta[-(burn.in)],plot=FALSE);t.eff <- 2*sum(autocor[[1]]) - 1;n.eff[2] <- nsteps/t.eff
autocor <- acf(mh$log_sigma[-(burn.in)],plot=FALSE);t.eff <- 2*sum(autocor[[1]]) - 1;n.eff[3] <- nsteps/t.eff
autocor <- acf(mh$mu_gamma[-(burn.in)],plot=FALSE);t.eff <- 2*sum(autocor[[1]]) - 1;n.eff[4] <- nsteps/t.eff
autocor <- acf(mh$log_sigma_gamma[-(burn.in)],plot=FALSE);t.eff <- 2*sum(autocor[[1]]) - 1;n.eff[5] <- nsteps/t.ef
n.eff
```

```
## [1] 1014.665 1086.073 1167.357 1172.466 1060.497
```

We check for correlation between the parameters by plotting graphs of a random sample of the iterations retained after discarding the burn-in:



The plots suggest that these parameters cannot all be taken to be independent in the posterior (e.g. there is a clear correlation between $\log(\alpha)$ and δ). The autocorrelation functions for these parameters are also not decaying as we would like. Due to the complexity of this sampler, it is quite difficult to determine how this may be remedied, so we instead consider an alternative approach based on numerical integration.

Given some realisation of the random variable $\boldsymbol{\theta}$, the marginal distribution of N_i may be recovered from the conditional distribution $N_i|\Gamma_i$ by integrating out Γ_i. This means that the likelihood of the data, $f(\mathbf{y}|\boldsymbol{\theta})$, is given by:

$$f(\mathbf{y}|\boldsymbol{\theta}) = \prod_{i=1}^{n} f(n_i|\boldsymbol{\theta}) = \prod_{i=1}^{n}\left[\int_0^{s_i} f(n_i|\gamma_i, \boldsymbol{\theta}) f(\gamma_i|\boldsymbol{\theta}) \mathrm{d}\gamma_i\right].$$

Code for evaluating this likelihood is given below. In particular, the function `integrand` evaluates the integrand for a given $(n_i, \gamma_i, \boldsymbol{\theta})$; `marginal` is a vector-valued function whose ith component is $f(n_i|\boldsymbol{\theta})$, calculated using `integrate`; and `likelihood` calculates $f(\mathbf{y}|\boldsymbol{\theta})$ using the product given above. A function which evaluates the prior distribution $f(\boldsymbol{\theta})$ is also supplied.

```
integrand <- function(gamma, N, s, censored, theta) {
  alpha <- exp(theta[1]); delta <- theta[2]; sigma <- exp(theta[3])
  mu_g <- theta[4]; sigma_g <- exp(theta[5])
  nshape <- 1/sigma; nscale <- alpha * (s - gamma)^delta; gshape <- 1/sigma_g; gscale <- exp(mu_g);
  conditional <- ifelse(censored, pweibull(N, shape=nshape, scale=nscale, lower.tail=FALSE),
                        dweibull(N, shape=nshape, scale=nscale))
  return(conditional * dweibull(gamma, shape=gshape, scale=gscale))
}

marginal <- Vectorize(function(N, s, censored, theta) {
  marginal <- integrate(integrand, lower=0, upper=s, N=N, s=s, censored=censored, theta=theta)
  return(marginal$value)
}, vectorize.args=c("N", "s", "censored"))

likelihood <- function(N, s, censored, theta) return(prod(marginal(N, s, censored, theta)))
prior <- function(theta) return(dexp(exp(theta[5]), rate=5))
```

Again, we use Metropolis-Hastings to sample from the posterior. However, we now use numerical integration to obtain the unconditional distribution of N_i, rather than sampling from the space of random effects. The implementation in `methast` allows for a Normal proposal distribution centred on the previous value of $\boldsymbol{\theta}$, as well as a Normal proposal with a fixed mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. The elements in $\boldsymbol{\Sigma}$ may also be scaled in such a way as to adjust the standard deviation of each component of $\boldsymbol{\theta}$, without changing the correlations between them - which gives us a way to tune the sampler in this case as well.

```
methast <- function(y, theta0, nsamples, scale=1, mean=NULL, cov=NULL) {
  N <- y$N; s <- y$s; censored <- y$ro == 1;
  p <- length(theta0); samples <- matrix(nrow=nsamples, ncol=p); scale <- rep_len(scale, p)
```

```
  theta.last <- theta0; posterior.last <- likelihood(N, s, censored, theta.last) * prior(theta.last)
  samples[1,] <- theta0; accepted <- 0

  fixedprop <- !(is.null(mean) || is.null(cov))
  if(fixedprop) { #all proposed thetas are from the same distribution
    covmat.scale <- Vectorize(function(i, j, cov, a) return(a[i]*a[j]*cov[i,j]), vectorize.args=c("i","j"))
    cov.scaled <- outer(1:p, 1:p, covmat.scale, cov=cov, a=scale) #scale the covariance matrix
    all.prop <- rmvnorm(nsamples-1, mean=mean, sigma=cov.scaled) #simulate all thetas now
    posterior.last <- posterior.last * dmvnorm(theta.last, mean=mean, sigma=cov.scaled)
  }

  for(i in 2:nsamples) {
    if(fixedprop) theta.prop <- all.prop[i-1,] else theta.prop <- rnorm(p, mean=theta.last, sd=scale)
    posterior.prop <- tryCatch(likelihood(N, s, censored, theta.prop) * prior(theta.prop),
                               error=function(cond) return("divergent integral"))
    if(posterior.prop == "divergent integral") { #for certain theta, integral may diverge
      samples[i,] <- theta.last; next #if so treat as a rejection
    }
    if(fixedprop) posterior.prop <- posterior.prop * dmvnorm(theta.prop, mean=mean, sigma=cov.scaled)

    alpha <- min(1, posterior.prop / posterior.last)
    if(alpha == 1 || runif(1) <= alpha) {
      theta.last <- theta.prop; posterior.last <- posterior.prop; accepted <- accepted + 1
    }
    samples[i,] <- theta.last
  }
  return(list(samples=samples, acceptrate=(accepted/nsamples)))
}
```

Before proceeding, it is worth noting that the calculation of $f(\mathbf{y}|\boldsymbol{\theta})$ requires the numerical integration of a 26-dimensional vector-valued function, so it takes some time for `metWe have discarded the first 1000 samples as burn-in.hast` to construct a Markov chain of the kind of length needed to be useful for Bayesian inference. If we are constructing correlated samples at each step, then this also has an adverse effect on performance.

We first construct a simple pilot chain to investigate any correlation between the parameters. A proposal standard deviation of 0.1 for all components of $\boldsymbol{\theta}$ is used here, and is sufficient to give an acceptance rate of close to 25%. The single standard deviation (as opposed to different values for each parameter) keeps the construction of a pilot chain straight-forward. Note that, even in this simple case, a reasonably long pilot chain is needed to ensure that the mean and variance of $\boldsymbol{\theta}|\mathbf{y}$ can be estimated sufficiently accurately.

```
set.seed(10000)
pilot <- methast(fatigue, c(1,1,1,1,1), 20000, scale=0.1); pilot$acceptrate; corr <- cor(pilot$samples); corr
```

```
## [1] 0.25165
```

```
##            [,1]       [,2]       [,3]       [,4]       [,5]
## [1,]  1.0000000 -0.9945154 -0.4590693 -0.2576392 -0.1954994
## [2,] -0.9945154  1.0000000  0.4351415  0.2542397  0.1849908
## [3,] -0.4590693  0.4351415  1.0000000  0.2413423  0.2286888
## [4,] -0.2576392  0.2542397  0.2413423  1.0000000  0.8001674
## [5,] -0.1954994  0.1849908  0.2286888  0.8001674  1.0000000
```

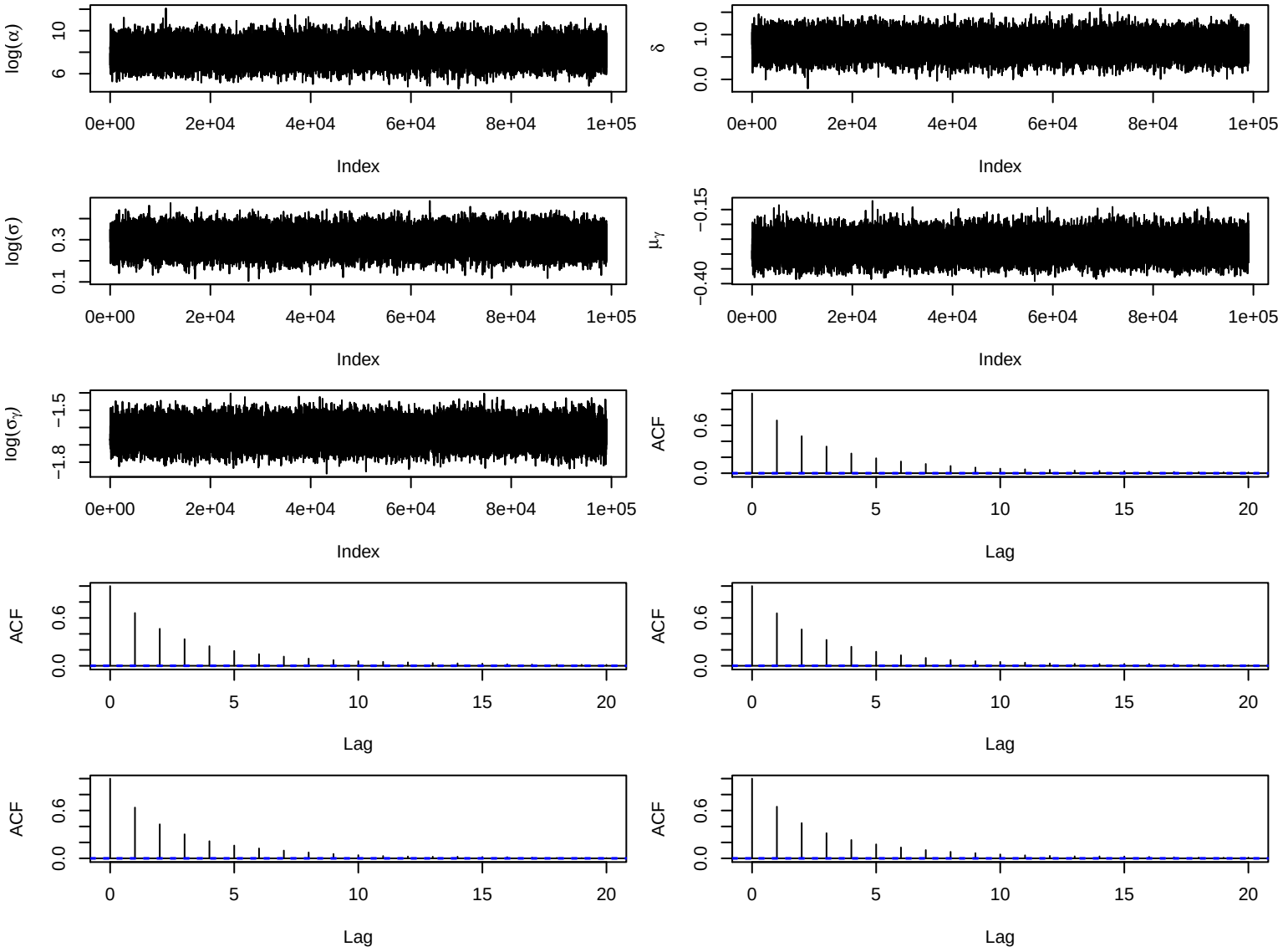
In particular, we note the existence of an almost perfect negative correlation between $\log(\alpha)$ and δ, and a very strong positive correlation between μ_γ and $\log(\sigma_\gamma)$. From the above it appears that all of the parameters are correlated to some non-negligible degree, which risks making the Metropolis-Hastings sampler very inefficient, in the sense that the resulting Markov chain would require an intolerably long time to converge to the limiting distribution (i.e. the posterior).

To resolve this, we take our full sample from the posterior using a modified Metropolis-Hastings algorithm; specifically, let $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\Sigma}}$ denote the estimated mean vector and covariance matrix, respectively, of $\boldsymbol{\theta}|\mathbf{y}$ (calculated using the pilot chain simulated above). Then, for the final sample from the posterior, we use the $N(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})$ as the proposal distribution.

```
set.seed(20000)
pilot.mean <- sapply(1:5, function(i, x) mean(x[,i]), x=pilot$samples); pilot.cov <- cov(pilot$samples)
posterior <- methast(fatigue, c(1,1,1,1,1), 100000, scale=0.3, mean=pilot.mean, cov=pilot.cov)
posterior$acceptrate
```

```
## [1] 0.27113
```

We have discarded the first 1000 samples as burn-in. To investigate the behaviour of the samples (and to try and verify that these samples are, indeed, drawn from $f(\boldsymbol{\theta}|\mathbf{y})$), we first the trace plots and autocorrelation functions below for each parameter. As explained in the previous section, we are keen to avoid excessive autocorrelation between the samples, as this will compromise the ability of the Markov chain to converge to its limiting distribution in a reasonable amount of time.



The trace plots show that the chain is mixing well; there are no points where there is a discernible change in the distribution of the samples, which suggests (albeit informally) that the chain has converged to its limiting distribution.

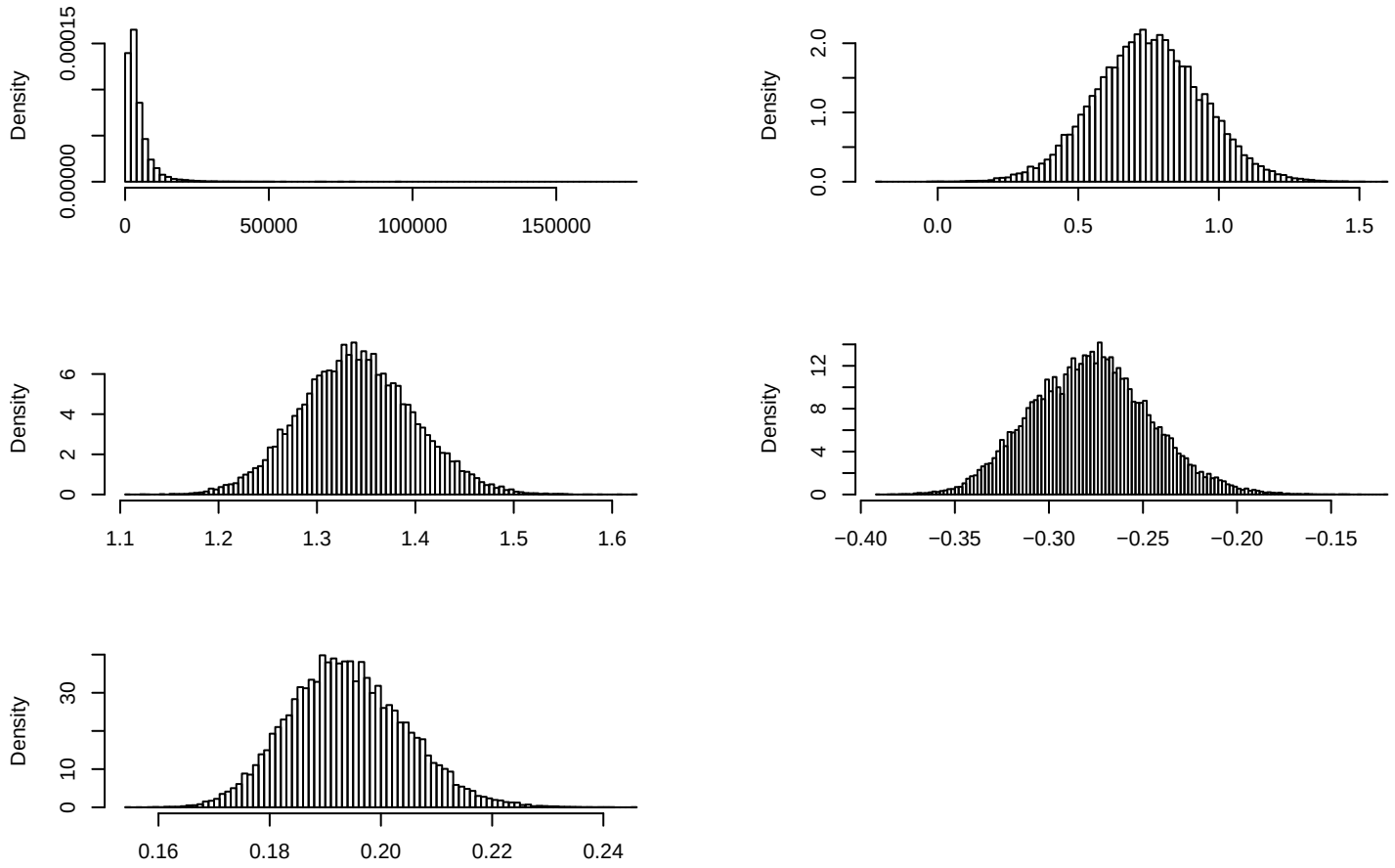
For all parameters, the first-order autocorrelation is almost perfect (as is to be expected, since the decision on whether to accept or reject a sampled value of $\boldsymbol{\theta}$ depends directly on the previously-sampled value). We also observe that, in all cases, the ACFs decay almost exponentially as the lag increases, and are almost equal to zero from lag 15 onwards. This offers further support to the claim that the distribution of the Markov chain has converged to the posterior.

To test this more formally, we take sub-samples for each parameter, and apply the two-sample Kolmogorov-Smirnov test for equality of distribution. For each parameter, we split the sample (after subsetting for burn-in) into two equal sizes, then sample every 20th value from each half to obtain two equally-sized sub-samples. As in the previous section, we use a two-sided test with the null hypothesis that the samples are drawn from the same distribution.

```
## [1] "p-value for log(alpha) = 0.903008367793445"
## [1] "p-value for delta = 0.482815729108645"
## [1] "p-value for log(sigma) = 0.597765025165641"
## [1] "p-value for mu[gamma] = 0.955977732335455"
## [1] "p-value for log(sigma[gamma]) = 0.342346593567781"
```

In all cases the p-values for the tests are much greater than 0.05 so in all cases we cannot, at the 5% significance level, reject the null hypothesis that the samples are drawn from the same distribution.

Taken together, these tests offer convincing empirical evidence that the Markov chain has converged to the posterior distribution. We now assume that this is indeed the case, and try to learn more about the posterior distribution using these sampled values. The below plots show histograms of the sampled values of each of the parameters:



The shape of the histogram for α is most striking, as it is highly positively skewed. The histograms for the the other parameters also clearly exhibit some features which suggest that $\boldsymbol{\theta}|\mathbf{y}$ does not follow a multivariate-Normal; all of them display varying degrees of asymmetry and skewness, for example.

Finally, 95% credible intervals for each of the parameters are given below.

```
##                2.5%     97.5%
## alpha       652.898 15641.618
## delta         0.376     1.122
## sigma         1.231     1.457
## mu_gamma     -0.336    -0.214
## sigma_gamma   0.175     0.216
```

Of particular interest is the extremely wide credible interval for α. This corresponds with the earlier analysis (in the frequentist setting), where the 95% confidence interval for the maximum likelihood estimate for $\log(\alpha)$ is also very wide.