Final ASI coursework

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.

```
library('survival'); library('eha')
rats <- read.table("http://people.bath.ac.uk/kai21/ASI/rats_data.txt") #load data
rx <- rats$rx; times <- rats$time; status <- rats$status; status <- as.logical(status)
```

Let T_i be the follow up time to tumor appearance in rat i. Assume the following probability model

$$T_i \sim \text{Weibull}(\text{shape} = 1/\sigma, \text{scale} = \exp(\eta_i)), \qquad \eta_i = \beta_0 + \beta_1 x_i, \quad x_i = \begin{cases} 1 & \text{rat } i \text{ received treatment} \\ 0 & \text{rat } i \text{ received control} \end{cases}$$

We assume that $\{T_1, \ldots, T_n\}$ are independent random variables.

Problem 1

We use `optim` 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))-sum(pweibull(times[!status],
}
#MLE with optim- Find the parameters that minimalise the negative log-likelihood, i.e. nll_weibull. Starting from
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
```

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

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

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

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

Problem 2

We compute a 95% asymptotic confidence interval for the treatment effect β_1. We use that $\hat{\theta} \sim N(\theta_t, \mathcal{I}^-1)$, where $\hat{\theta}$ is our MLE, θ_t is the vector of the true values and $\mathcal{I}^{-1}$ is the information matrix. We approximate the information matrix by the Hessian $\nabla^2 l(\hat{\theta})$, where l is the log-likelihood. To get the confidence interval for β_1, we standardise our ML estimate $\hat{\beta}_1$:

$$\mathbb{P}(\beta_{1,t} - x < \hat{\beta}_1 \leq \beta_{1,t} + x) = \mathbb{P}\left(-\frac{x}{\sigma_{\beta_1}} < \frac{\hat{\beta}_1 - \beta_{1,t}}{\sigma_{\beta_1}} \leq \frac{x}{\sigma_{\beta_1}}\right) = 2\Phi\left(\frac{x}{\sigma_{\beta_1}}\right) - 1 = 0.95,$$

where $\beta_{1,t}$ denotes the true value of β_1 and Φ is the cdf of the standard normal distribution. From the equation above we get $x = 1.96\sigma_{\beta_1}$.

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

```
## [1] -0.41311800 -0.06390767
```

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, so this treatment should not be used.

Problem 3

We now assume that

$$T_i \sim \text{log-logistic}(\text{shape} = 1/\sigma, \text{scale} = \exp(\eta_i)).$$

We find the maximum likelihood estimate of the parameter $\boldsymbol{\theta}^\top = (\beta_0, \beta_1, \log(\sigma))$ using `optim` as well as the asymptotic standard errors of each parameter estimate. We also compute a 95% asymptotic confidence interval for the treatment effect β_1.

```
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]
}
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) #ml
fit_llogis$convergence #check convergence
```

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

```
fit_llogis$par #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$hessian))^.5 #standard errors
c(fit_llogis$par[2]-1.96*se_llogis[2],fit_llogis$par[2]+1.96*se_llogis[2])#confidence interval for beta1
```

```
## [1] -0.41689768 -0.04211626
```

Interpretation: The confidence interval shows again that with probability 95% β_1 has a slightly negative effect, so this treatment should not be used.

Problem 4

Now assume we model the effect of the litters as random effects as follows:

$$T_i \sim \text{Weibull}(\text{shape} = 1/\sigma, \text{scale} = \exp(\eta_i)), \qquad \eta_i = \beta_0 + \beta_1 x_i + b_{litter(i)}$$

where $litter(i) \in \{1, 2, \ldots, 49, 50\}$ indicates the litter to which rat i belogs to. Assume that the random effects $\{b_1, \ldots, b_{50}\}$ are iid $N(0, \sigma_b^2)$ where σ_b^2 is unknown. Note that now we have $\boldsymbol{\theta} = (\beta_0, \beta_1, \log(\sigma), \log(\sigma_b))^\top$.

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 $\mathbf{b} \sim N(\mathbf{0},$ and using the Normal pdf, $\sigma_b^2\mathbf{I})$, $\log[f(\mathbf{b}|\boldsymbol{\theta})] = -\sum_{j=1}^{n_b}\{\log(\sigma_b) + \log(2\pi)/2 + b_j^2/2\sigma_b^2\}$. 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.nll$par, -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
c(mle.b1 - 1.96*sd.mle.b1, mle.b1 + 1.96*sd.mle.b1) #95% CI
```

```
## [1] -0.4033757 -0.0582141
```

Under the assumptions made here, the above interval contains the "true" value of β_1 under this model with 95% probability. The interval does not contain zero (and in fact contains no non-negative numbers at all). Therefore, we conclude that the value of η, and of the shape parameter in the distribution of T, is lower for a rat that received the treatment compared to an otherwise identical rat that received the placebo.

Problem 5

We use a random walk Metropolis-Hastings algorithm to sample from the posterior distribution of $\theta = (\beta_0, \beta_1, \log(\sigma), \log(\sigma_b))^T$ using the model above. We follow a Bayesian estimation procedure and specify the following prior distributions on the unknown parameters: β_0 β_1 and $\log(\sigma)$ are independent and following uniform (improper) priors while σ_b is also independent of β_0 β_1 and $\log(\sigma)$ and follows a prior exponential distribution with rate 5. 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_pi0_b_tumour=log(dnorm(b_tumour, mean=0, sd=exp(log_sigma_b)));  log_pio_b_dead=log(dnorm(b_dead, mean=0, s
  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_tu
  log.lik<-sum(log (dweibull(tumour$time, 1/exp(log_sigma), exp(beta0+beta1*tumour$rx+b_tumour))))+sum(log (pweibu
  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 β_0, β_1, $\log(\sigma_b)$, $\log(sigma)$ and each element of b I will also use a symmetric multivariate normal distribution centered on the current values for the values of b. I 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 as otherwise, that is, 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. Thus we choose as our initial conditions the maximum likelihood estimators calculated earlier.

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

```
split_rats <- split(rats, rats$status)
tumour=as.data.frame(split_rats[[2]]); dead=as.data.frame(split_rats[[1]])
nsteps=100000 ; burn.in=10000
MH<-function(beta0_0,beta1_0, log_sigma_0, log_sigma_b0, b_tumour0, b_dead0, sigma_beta0, sigma_beta1, sigma_log_s
  accept <- rep(0,3) # set up locations to store values at each step
  beta0 <- rep(0,nsteps) ;  beta1=rep(0,nsteps);  log_sigma=rep(0,nsteps);  log_sigma_b=rep(0,nsteps)
  b_tumour=matrix(0,nsteps,length(tumour$rx)) ;  b_dead=matrix(0,nsteps,length(dead$rx)) # set up locations to sto
  beta0[1] <- beta0_0 ;  beta1[1]=beta1_0;  log_sigma[1]=log_sigma_0;  log_sigma_b[1]=log_sigma_b0;  b_tumour[1,]=
lp0 <- log.post(beta0_0,beta1_0, log_sigma_0, log_sigma_b0,b_tumour0, b_dead0,tumour, dead) # calculate log poster
for( i in 2:nsteps){ #MH loop
current_beta0=beta0[i-1] ;current_beta1=beta1[i-1];current_log_sigma=log_sigma[i-1];  current_log_sigma_b=log_sigma
proposed_log_sigma_b=current_log_sigma_b+rnorm(1,0,sigma_log_sigma_b)#update sigma_b
lp1 <- log.post(current_beta0,current_beta1, current_log_sigma, proposed_log_sigma_b,current_b_tumour, current_b_d
acc <- exp(min(0,lp1-lp0))
if (runif(1)>=acc| !is.finite(acc)){#reject
  b_tumour[i,] <- current_b_tumour ;  b_dead[i,]=current_b_dead;  beta0[i] <- current_beta0;  beta1[i]=current_beta
  lp1<- lp0 ## Return to the 'old' log posterior
}else {#accept
  accept[1]=accept[1]+1 # keep track of number of acceptances
  log_sigma_b[i]=proposed_log_sigma_b #store found values
  lp0 <- lp1 ## uldate old log posterior to the new one
    proposed_b_tumour=current_b_tumour + rnorm( length(tumour$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, propose
  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_beta
  lp1<- lp0 ## Return to previous log posterior
```

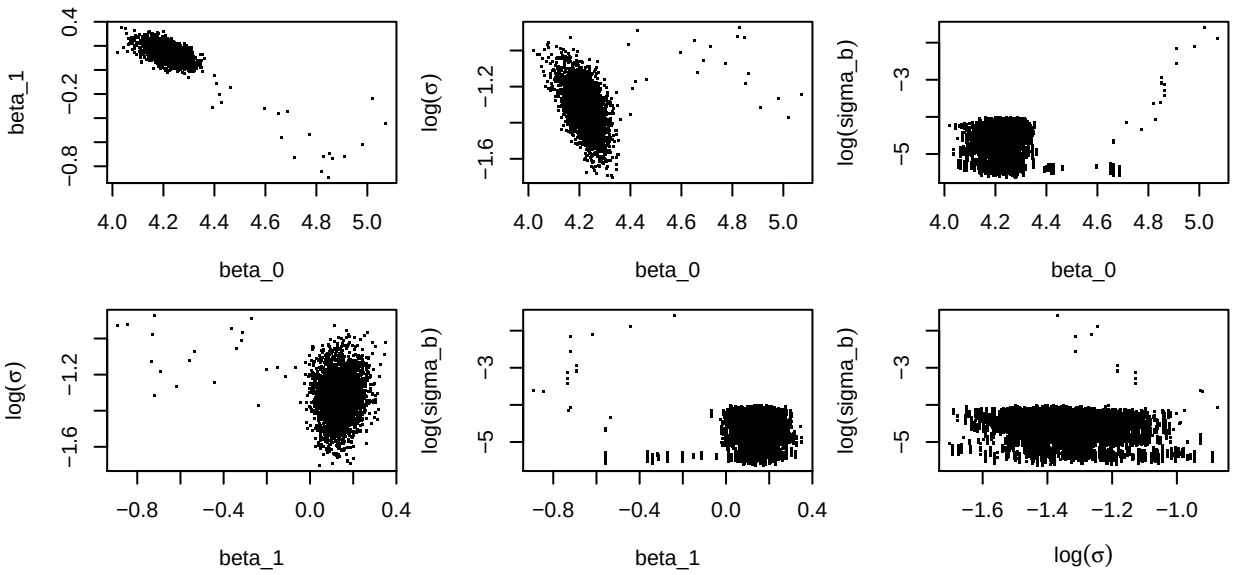
```
}else {#accept
          accept[2]=accept[2]+1 # keep track to calculate acceptance rates
          b_tumour[i,] <- proposed_b_tumour;  b_dead[i,]=proposed_b_dead;  #store values
          lp0 <- lp1 ## update log posterior
  proposed_beta0=current_beta0+rnorm(1,0,sigma_beta0); proposed_beta1=current_beta1+rnorm(1,0,sigma_beta1)
  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_
  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=
}
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
```

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] 0.3343400 0.4926123 0.1700668
```

We check for correlation between the parameters by plotting graphs.



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
proposed_beta0=proposed[1];  proposed_beta1=proposed[2];  proposed_log_sigma=proposed[3]
```

Using the new MH algorithm and tuning we get

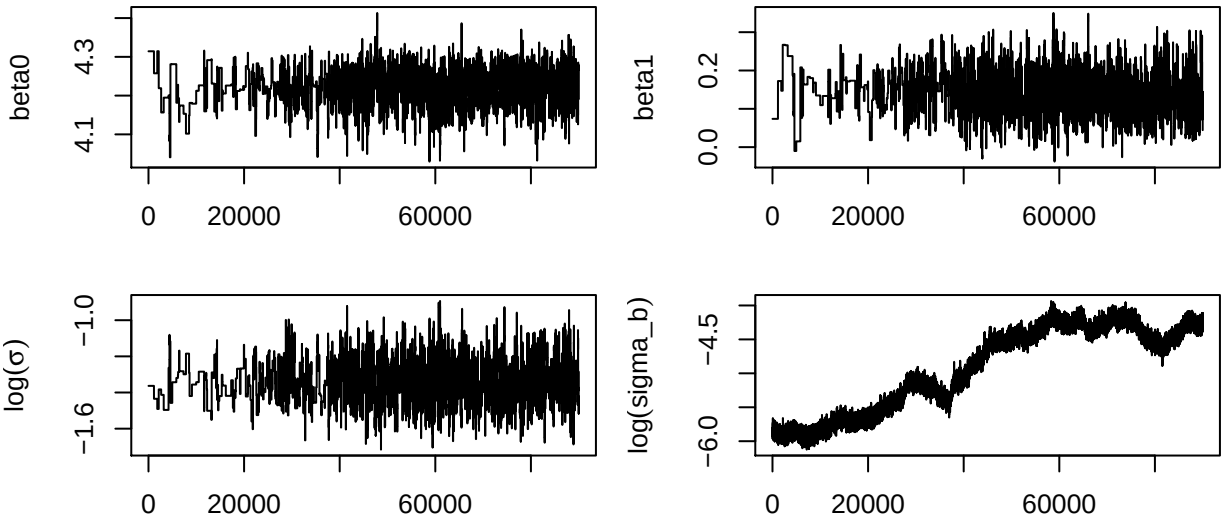
```
mh=MH2(5.0188886, -0.2376741, -1.3687252,-1.5976068, b_tumour0 =  rep(0, length(tumour$rx)),b_dead0= rep(0, length
mh$ar_outer;mh$ar_middle;mh$ar_inner
```

```
## [1] 0.33399
```

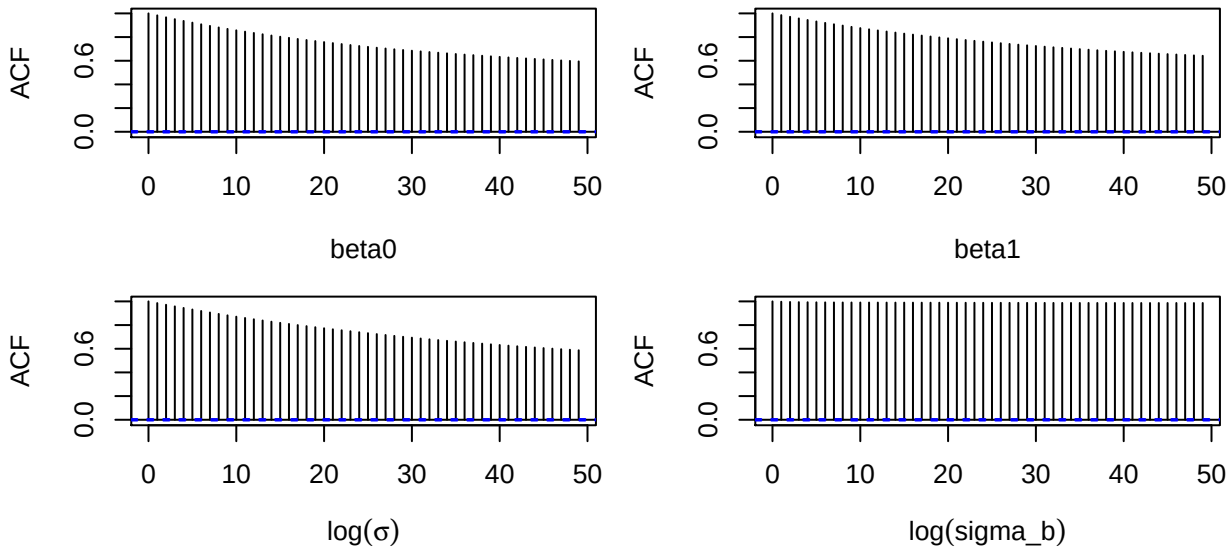
```
## [1] 0.366478
```

```
## [1] 0.2119281
```

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] 1356.322 1301.326 1342.889 1021.657
```

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 chance to see that they come from the same distribution and hence that the Markov chain has converged.

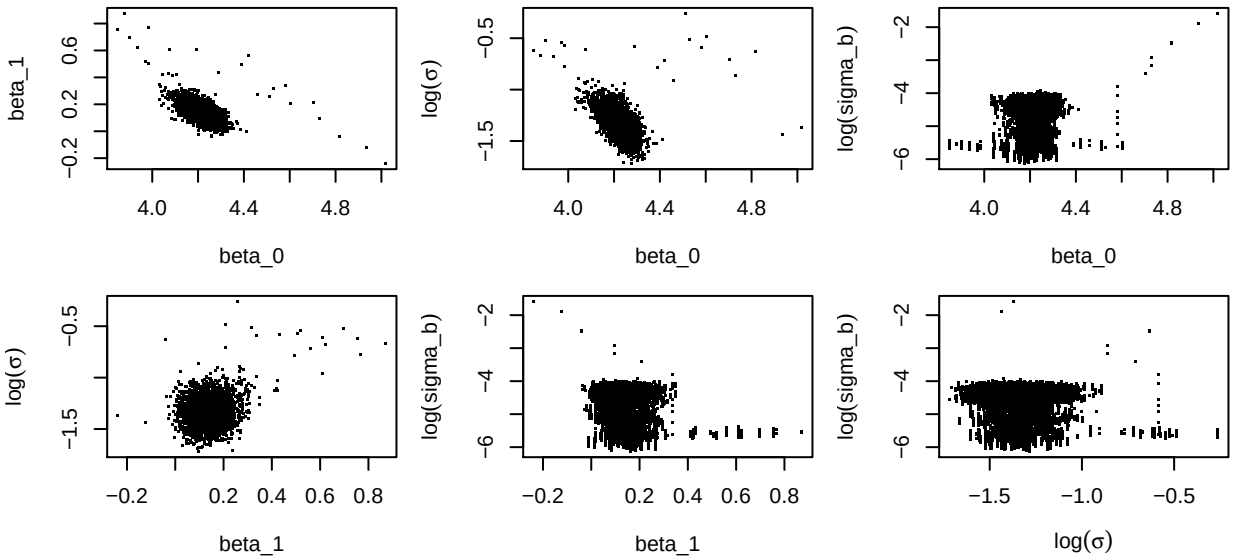
```
k1=ks.test(mh$beta0[-1000:-500], mh$beta0[-500]);
k2=ks.test(mh$beta1[-1000:-500], mh$beta1[-500]);
k3=ks.test(mh$log_sigma_b[-1000:-500], mh$log_sigma_b[-500]);
```

```
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] 0.1813060 0.1681842 0.6700861 0.1961523
```

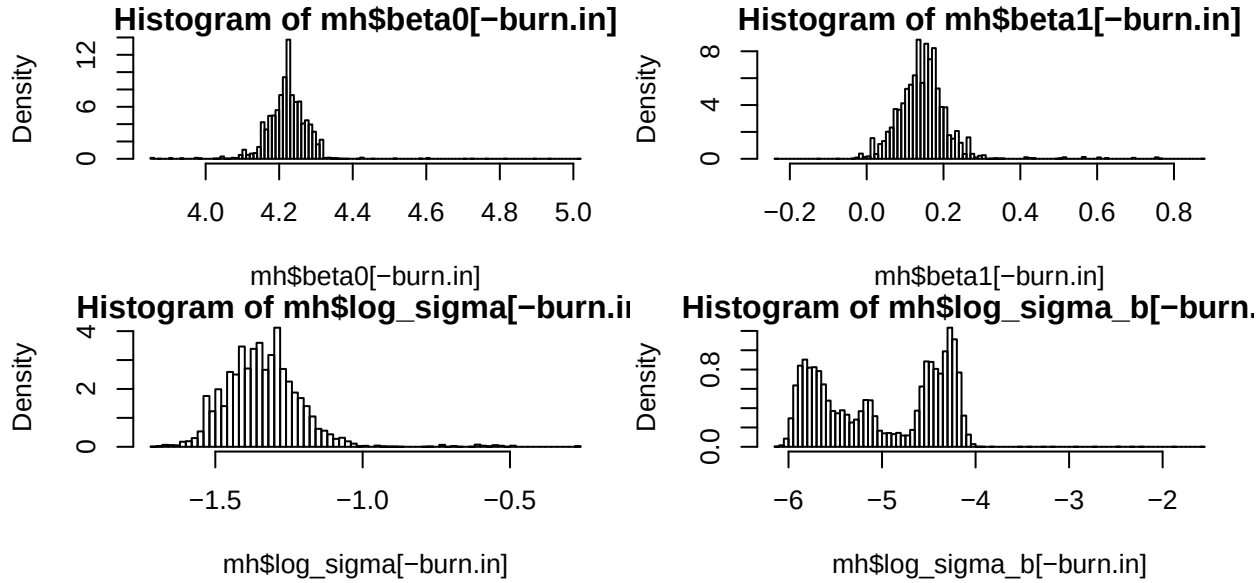
We see that in all cases the p-values are large, and thus there is not significant evidence that the two sub-samples are from different distributions. Hence we conclude that the Markov chain has converged.

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%
## 0.02342388 0.26707381
```

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 confidence values for β_1 being greater than zero suggest that the treatment is having a poitive effect. This is in contrast to previous results and thus perhaps suggests that more investigation is required.

Fatigue of materials

The fatigue characteristics of materials are established through testson small flat plates called coupons. The levels of stress (force per unit area) as well as the number of cycles to failure are recorded for each test. The number of cycles to failure N_i exhibits a random behaviour due to inherent microstructural inhomogeneity in the material properties and also due to uncontrolled differences in test conditions. When a test is stopped before the coupon fails, then the coupon is marked as a *runout*. Let N_i denote the number cycles until failure in coupon i and s_i be the corresponding stress level (in Mega Pascals) applied.
We model N_i as continuous using $N_i = \alpha\,(s_i - \gamma)^{\delta}\,\epsilon_i\,,$ where $s_i > \gamma$ and ϵ_i is a random error such that $\epsilon_i \sim$ Weibull(shape $= 1/\sigma$, scale $= 1$) . The constants $\alpha > 0$, $\delta \in R$, $\gamma > 0$ and $\sigma > 0$ are unknown parameters. Empirical results suggest that coupons tested below the stress level γ will never fail. The unknown parameter γ is therefore called the **fatigue limit.** Consider the following data obtained in a series of 26 tests to study the fatigue of a nickel base supperalloy:

```
fatigue <- read.table("http://people.bath.ac.uk/kai21/ASI/fatigue.txt")#load data
s <- fatigue$s; N <- fatigue$N; ro <- fatigue$ro; ro <- as.logical(ro)
```

Problem 1

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

```
## [1] 13.572230 -1.045889 -0.510714
```

```
-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$hessian))^.5 #standard deviations
c(fit_weibull2$par[1] - 1.96*se_weibull2[1], fit_weibull2$par[1] + 1.96*se_weibull2[1]) #confidence limit for loga
```

```
## [1] 12.74092 14.40354
```

```
c(fit_weibull2$par[2 ]- 1.96*se_weibull2[2], fit_weibull2$par[2] + 1.96*se_weibull2[2]) #confidence limit for delt
```

```
## [1] -1.3424133 -0.7493654
```

```
c(fit_weibull2$par[3] - 1.96*se_weibull2[3], fit_weibull2$par[3] + 1.96*se_weibull2[3]) #confidence limit for logs
```

```
## [1] -0.8668891 -0.1545389
```

The function `gamma_sensitivity`,code is not displayed, calls `optim` for a given initial point θ_0 and for several values of γ. 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))]
```

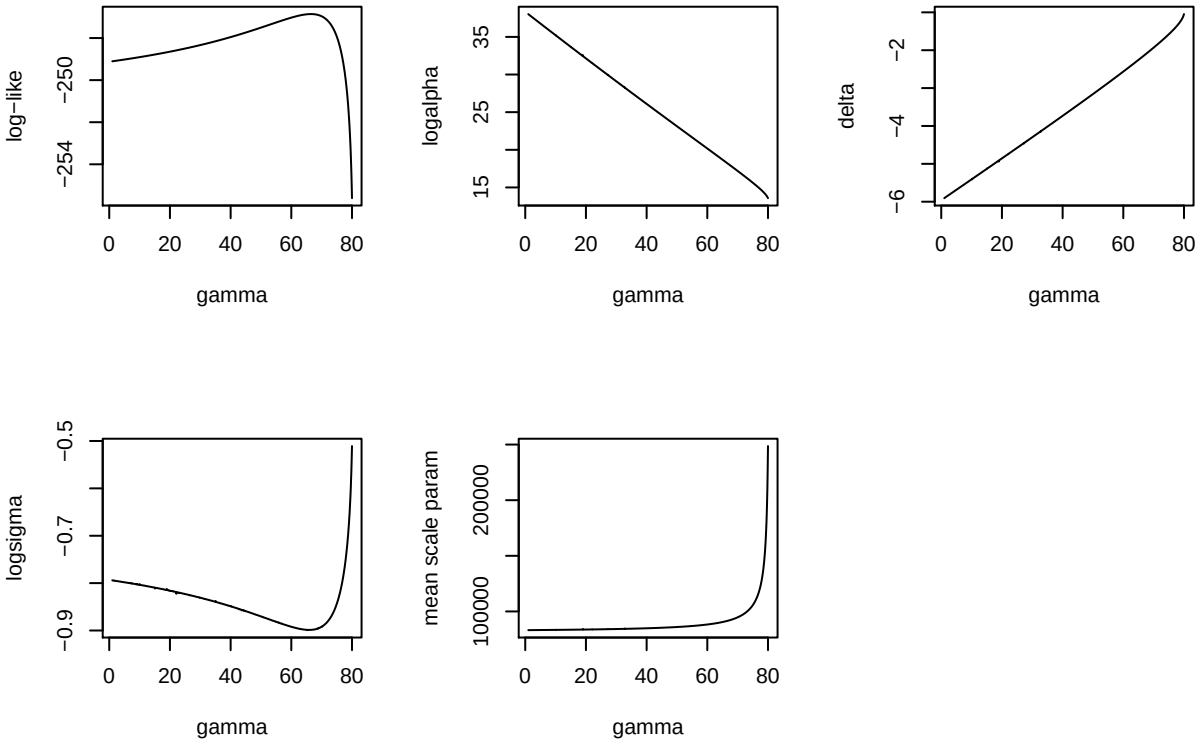
```
## [1] -2.160232
```

```
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 very same shape as $\log(\sigma)$. This is in accordance with 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. About the shape parameter: $\log(\sigma)$ has a minimum (and the shape parameter a maximum) at γ=65.2. As long as the (mean of the) scale parameter changes moderately, the decrease can be explained as following: When γ is small the minimum stress level is much higher than the fatigue limit. Therefore early failures are more likely even in case of stress levels from the lower region of the data. 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 the failure to occure for stress levels close to the limit. This gives smaller positive skewness, and bigger shape parameter (smaller $\log(\sigma)$). As soon as the mean scale parameter starts to increase very fast (around γ>66) this explanation fails, and positive skewness occurs again. About 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 opposite effects of γ on α and δ keep the scale parameter in the same order of magnitude for different γ values (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).

Problem 2

We estimate the vector of unknown parameters. Now γ is included in the parameter vector θ, and has the constraint $0 < \gamma < \min(s[!ro]) = \min(s)$ in this case. So 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)) - sum(pweibull(N[ro], shape
}
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. t
fit_weibull2_gam <- optim(par=theta0, fn=nll_weibull2_gam, method="BFGS", hessian=TRUE, s=s, N=N, ro=ro) #mle with
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 the followings:

```
## [1] 12.93626 23.54776
```

```
## [1] -3.309305 -1.010041
```

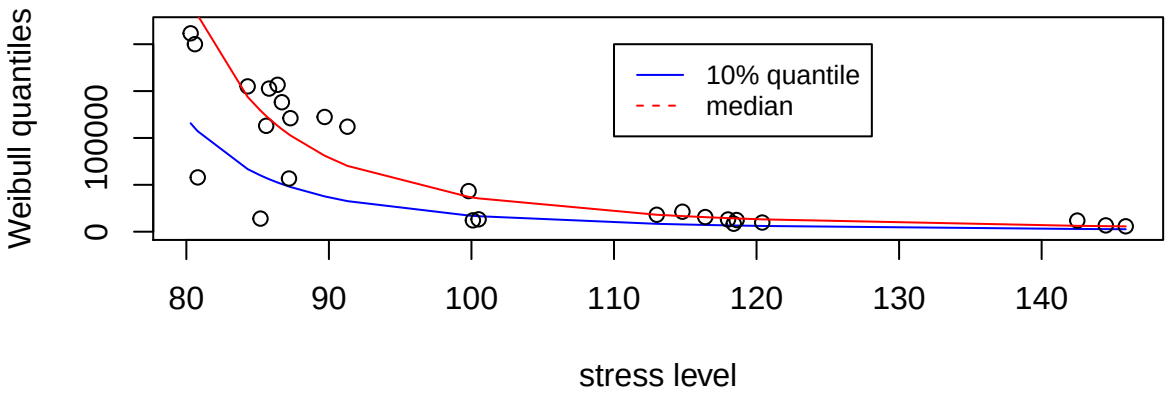
```
## [1] -1.2431432 -0.5541681
```

```
## [1] 0.05152919 3.09497714
```

Problem 3

We 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
```



Problem 4

Consider now the following 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. For coupon i, the conditional distribution of the number of cycles to

failure N_i given that $\Gamma_i = \gamma_i < s_i$ that is, given that the realization of the fatigue limit for that coupon is below the applied stress level is given by

$$N_i|\Gamma_i = \gamma_i < s_i \sim \text{Weibull}(\text{shape} = 1/\sigma, \text{scale} = \alpha\,(s_i - \gamma_i)^{\delta}) \tag{1}$$

We will assume that $\Gamma_1, \ldots, \Gamma_{26}$ are i.i.d:

$$\text{Weibull}(\text{shape} = 1/\sigma_\gamma, \text{scale} = \exp(\mu_\gamma)) \tag{2}$$

where $\mu_\gamma \in R, \sigma_\gamma > 0$ are unknown parameters. Let $b = (\gamma_1, \ldots, \gamma_{26})^\top$ and the vector of unknown parameters is now given by $\theta^\top = (\log(\alpha), \delta, \log(\sigma), \mu_\gamma, \log(\sigma_\gamma))$. We will assume the following priors: $\log(\alpha), \delta, \log(\sigma), \mu_\gamma$ are independent and with improper uniform priors while σ_γ is exponential with rate 5 and independent of $\log(\alpha), \delta, \log(\sigma)$ and μ_γ.

We wish to use a Metropolis-Hastings algorithm to sample from the posterior distribution of θ and the random effects 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
  log.lik=sum(log(pweibull(broke$N, 1/exp(log_sigma), exp(log_alpha)*(broke$s-gamma_broke)^delta)))+sum(log(dweibul
  return(log.lik+log_pi0) # now the log posterior = log likelihood +log prior
}
```

We now move onto the MH algorithm. for the proposal distributions we consider a normal random variable centered on the previous value with variance that we can tune for the variables $\log(\alpha), \delta, \log(\sigma), \mu_\gamma$ and $\log(\sigma_\gamma)$ and a uniform random variable centered on the previous value with range that we can vary for γ. We choose a burn in period of 10000 although this can be altered and 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.

```
split_fatigue <- split(fatigue, fatigue$ro)
broke=as.data.frame(split_fatigue[[1]]); runoff=as.data.frame(split_fatigue[[2]])
nsteps=100000; burn.in=10000
MH<-function(log_alpha0, delta0, log_sigma0,mu_gamma0,log_sigma_gamma0, gamma_broke0, gamma_runoff0, range_log_alph
  accept <- rep(0,3)# will keep track of acceptance rates for the inner middle and outer metropolis hastings chain
  log_alpha <- rep(0,nsteps); delta=rep(0,nsteps);log_sigma=rep(0,nsteps); mu_gamma=rep(0,nsteps);  log_sigma_gamma
  gamma_broke=matrix(0, nsteps, length(broke$N)); gamma_runoff=matrix(0, nsteps, length(runoff$N))#Set up holders
  log_alpha[1] <- log_alpha0;  delta[1]=delta0;  log_sigma[1]=log_sigma0;  mu_gamma[1]=mu_gamma0;  log_sigma_gamma
  gamma_broke[1,]=gamma_broke0;  gamma_runoff[1,]=gamma_runoff0#Set initial values
  lp0 <- log.post(log_alpha0, delta0, log_sigma0,mu_gamma0,log_sigma_gamma0, gamma_broke0, gamma_runoff0, broke, ru
  for (i in 2:nsteps){
    #Set current values
    current_log_alpha=log_alpha[i-1]; current_delta=delta[i-1]; current_log_sigma=log_sigma[i-1]; current_mu_gamma=
    current_gamma_broke=gamma_broke[i-1,];current_gamma_runoff=gamma_runoff[i-1,] #extract current values
    proposed_mu_gamma=current_mu_gamma+rnorm(1,0, range_mu_gamma) # new proposed values
    proposed_log_sigma_gamma=current_log_sigma_gamma+rnorm(1,0, range_log_sigma_gamma) # new proposed values
    lp1 <- log.post(current_log_alpha, current_delta, current_log_sigma,proposed_mu_gamma,proposed_log_sigma_gamma
        acc <- exp(min(0,lp1-lp0))
    if (runif(1) >= acc | !is.finite(acc)){# reject
      log_alpha[i] <- current_log_alpha; delta[i]=current_delta; log_sigma[i]=current_log_sigma;    mu_gamma[i]=cu
      lp1=lp0 # keep log posterior values up to date
    }else{#accept
      accept[1]=accept[1]+1 # keep track to calculate acceptance rates
      mu_gamma[i]=proposed_mu_gamma ; log_sigma_gamma[i]=proposed_log_sigma_gamma # update new values
      lp0=lp1# keep log posterior values up to date
      proposed_gamma_broke= current_gamma_broke + runif(length(broke$N), -range_gamma, range_gamma)# propose new g
      proposed_gamma_runoff= current_gamma_runoff+ runif(length(runoff$N), -range_gamma, range_gamma) # propose ne
      check1=isTRUE(all.equal(abs(broke$s-proposed_gamma_broke), broke$s-proposed_gamma_broke)) #check gamma is le
      check2=isTRUE(all.equal(abs(runoff$s-proposed_gamma_runoff), runoff$s-proposed_gamma_runoff))#check gamma is
      if(!(check1 & check2) ){ # reject
      log_alpha[i] <- current_log_alpha;delta[i]=current_delta;log_sigma[i]=current_log_sigma; gamma_broke[i,]=cur
      lp1=lp0# keep log posterior values up to date
```

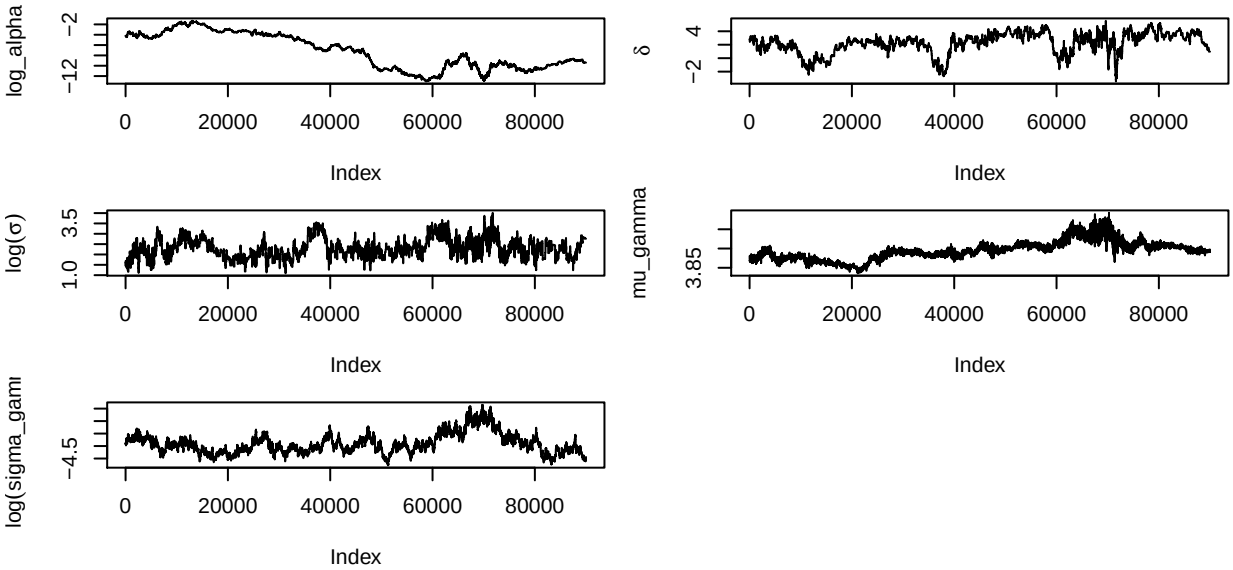
```
      } else{# continue with MH
      lp1 <- log.post(current_log_alpha, current_delta, current_log_sigma,proposed_mu_gamma,proposed_log_sigma_gamm
      acc <- exp(min(0,lp1-lp0))
      if (runif(1) >= acc| !is.finite(acc)){# reject
      log_alpha[i] <- current_log_alpha;delta[i]=current_delta; log_sigma[i]=current_log_sigma; gamma_broke[i,]=cu
      lp1=lp0# keep log posterior values up to date
      }else{ #accept
        accept[2]=accept[2]+1 # keep track to calculate acceptance rates
        gamma_broke[i,]=proposed_gamma_broke; gamma_runoff[i,]=proposed_gamma_runoff # update new values
      lp0=lp1# keep log posterior values up to date
      proposed_log_alpha=current_log_alpha+rnorm(1, 0, range_log_alpha) #new propsed values
      proposed_delta=current_delta+rnorm(1, 0, range_delta) #new propsed values
      proposed_log_sigma=current_log_sigma+rnorm(1, 0, range_log_sigma) #new propsed values
      lp1 <- log.post(proposed_log_alpha, proposed_delta, proposed_log_sigma,proposed_mu_gamma,proposed_log_sigma_g
      acc <- exp(min(0,lp1-lp0))
      if (runif(1) >= acc| !is.finite(acc)){# reject
      log_alpha[i] <- current_log_alpha; delta[i]=current_delta;log_sigma[i]=current_log_sigma # keep track of var
      lp1=lp0# keep log posterior values up to date
      }else{#accept
        accept[3]=accept[3]+1 # keep track to calculate acceptance rates
        log_alpha[i] <- proposed_log_alpha
      delta[i]=proposed_delta ; log_sigma[i]=proposed_log_sigma # keep track of variables
      lp0=lp1# keep log posterior values up to date
      } } }  }
list(log_alpha=log_alpha, delta=delta, log_sigma=log_sigma,mu_gamma=mu_gamma,log_sigma_gamma=log_sigma_gamma, ar_i
}
  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] 0.10631
```

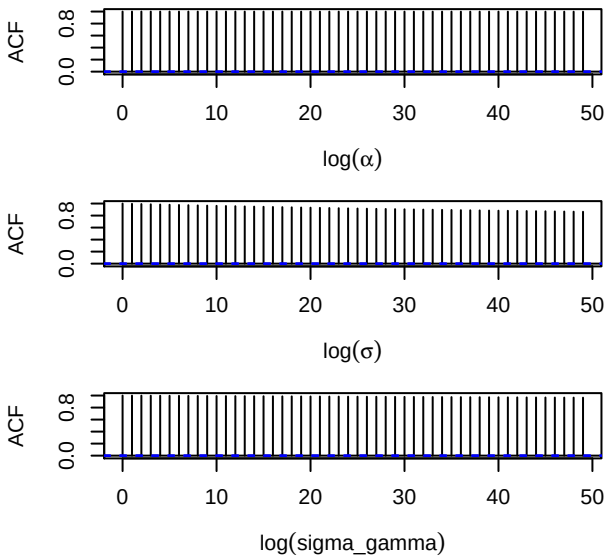
```
## [1] 0.6138651
```

```
## [1] 0.582133
```

We plot the trace graphs of our retained results minus the burn-in period:



and the auto-correlation 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[4] <- nsteps/t.ef
n.eff
```

```
## [1] 1011.017 1027.229 1095.069 1028.925    0.000
```

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