

Fatigue of materials - part

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

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

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

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

where $\mu_\gamma \in R, \sigma_\gamma > 0$ are unknown parameters. Let $b = (\gamma_1, \dots, \gamma_{26})^\top$ 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 . For an initial step we split the data set into those in which the coupon broke and those which didn't.

```
split_fatigue <- split(fatigue, fatigue$ro)
broke=as.data.frame(split_fatigue[[1]])
runoff=as.data.frame(split_fatigue[[2]])
```

The below computes a log posterior function for the model given parameters: $\log(\alpha), \delta, \log(\sigma), \mu_\gamma, \log(\sigma_\gamma)$ and γ

```
# function to compute log posterior
log.post <- function(log_alpha, delta, log_sigma, mu_gamma, log_sigma_gamma, gamma_broke, gamma_runoff, br
  log_pi0_log_alpha=log(1);log_pi0_delta=log(1);log_pi0_log_sigma=log(1);log_pi0_mu_gamma=log(1) #lo
  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
  log.lik=sum(log(pweibull(broke$N, 1/exp(log_sigma), exp(log_alpha)*(broke$s-gamma_broke)^delta))+sum
  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.

```
nsteps=100000; burn.in=10000
MH<-function(log_alpha0, delta0, log_sigma0, mu_gamma0, log_sigma_gamma0, gamma_broke0, gamma_runoff0, ran
  accept <- rep(0,3)# will keep track of acceptance rates for the inner middle and outer metropolis has
  log_alpha <- rep(0,nsteps); delta=rep(0,nsteps);log_sigma=rep(0,nsteps); mu_gamma=rep(0,nsteps); log
  gamma_broke=matrix(0, nsteps, length(broke$N)); gamma_runoff=matrix(0, nsteps, length(runoff$N))#Set
  log_alpha[1] <- log_alpha0; delta[1]=delta0; log_sigma[1]=log_sigma0; mu_gamma[1]=mu_gamma0; log_
```

```

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)
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_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,proposed_gamma_broke,proposed_gamma_runoff)
  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]=current_mu_gamma; gamma_broke[i,]=current_gamma_broke; gamma_runoff[i,]=current_gamma_runoff
    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$N), -range_gamma, range_gamma)# proposed values
    proposed_gamma_runoff= current_gamma_runoff+ runif(length(runoff$N), -range_gamma, range_gamma) # proposed values
    check1=isTRUE(all.equal(abs(broke$s-proposed_gamma_broke), broke$s-proposed_gamma_broke)) #check 1
    check2=isTRUE(all.equal(abs(runoff$s-proposed_gamma_runoff), runoff$s-proposed_gamma_runoff))#check 2
    if(!(check1 & check2) ){ # reject
      log_alpha[i] <- current_log_alpha;delta[i]=current_delta;log_sigma[i]=current_log_sigma; gamma_broke[i,]=current_gamma_broke; gamma_runoff[i,]=current_gamma_runoff
      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_gamma,proposed_gamma_broke,proposed_gamma_runoff)
      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,]=current_gamma_broke; gamma_runoff[i,]=current_gamma_runoff
        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 proposed values
        proposed_delta=current_delta+rnorm(1, 0, range_delta) #new proposed values
        proposed_log_sigma=current_log_sigma+rnorm(1, 0, range_log_sigma) #new proposed values
        lp1 <- log.post(proposed_log_alpha, proposed_delta, proposed_log_sigma,proposed_mu_gamma,proposed_log_sigma_gamma,proposed_gamma_broke,proposed_gamma_runoff)
        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 variables
          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, gamma_broke=gamma_broke, gamma_runoff=gamma_runoff)
}

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)
mh$ar_outer; mh$ar_middle; mh$ar_inner

```

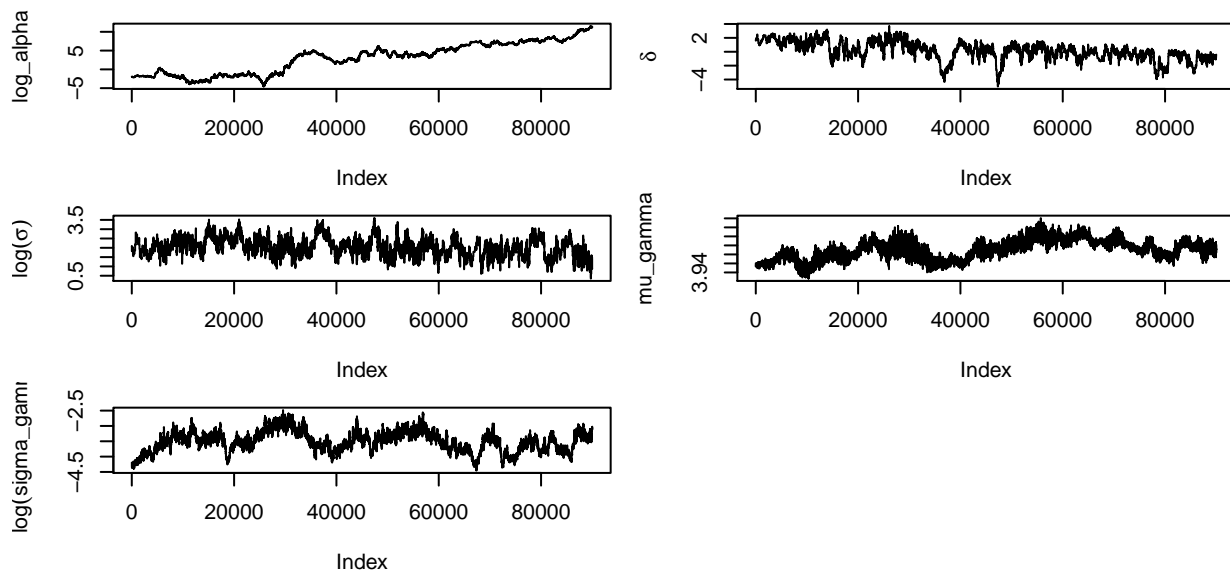
```
## [1] 0.1411
```

```
## [1] 0.7229624
```

```
## [1] 0.5485737
```

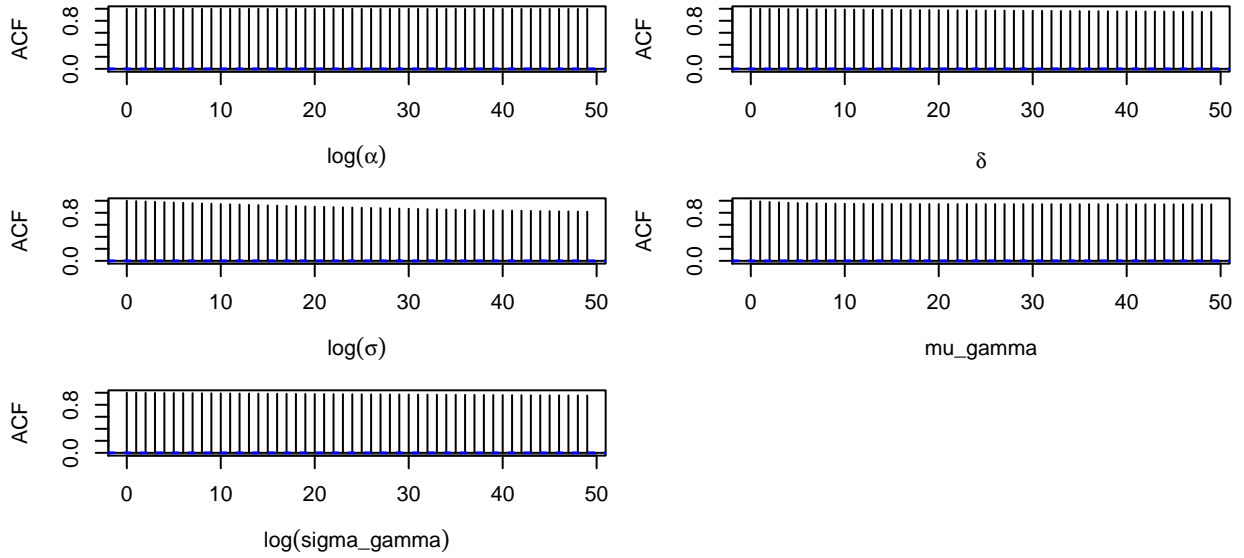
We plot the trace graphs of our retained results minus the burn-in period:

```
show.plot<- (burn.in):nsteps
par(mfrow=c(3,2), mar=c(4,4,1,1))
plot(mh$log_alpha[show.plot],type="l",ylab=expression(log_alpha))
plot(mh$delta[show.plot],type="l",ylab=expression(delta))
plot(mh$log_sigma[show.plot],type="l",ylab=expression(log(sigma)))
plot(mh$mu_gamma[show.plot],type="l",ylab=expression(mu_gamma))
plot(mh$log_sigma_gamma[show.plot],type="l",ylab=expression(log(sigma_gamma)))
```



and the auto-correlation functions:

```
par(mfrow=c(3,2),mar=c(4,4,1,1))
acf(mh$log_alpha[-burn.in],xlab=expression(log(alpha)))
acf(mh$delta[-burn.in],xlab=expression(delta))
acf(mh$log_sigma[-burn.in],xlab=expression(log(sigma)))
acf(mh$mu_gamma[-burn.in],xlab=expression(mu_gamma))
acf(mh$log_sigma_gamma[-burn.in],xlab=expression(log(sigma_gamma)))
```

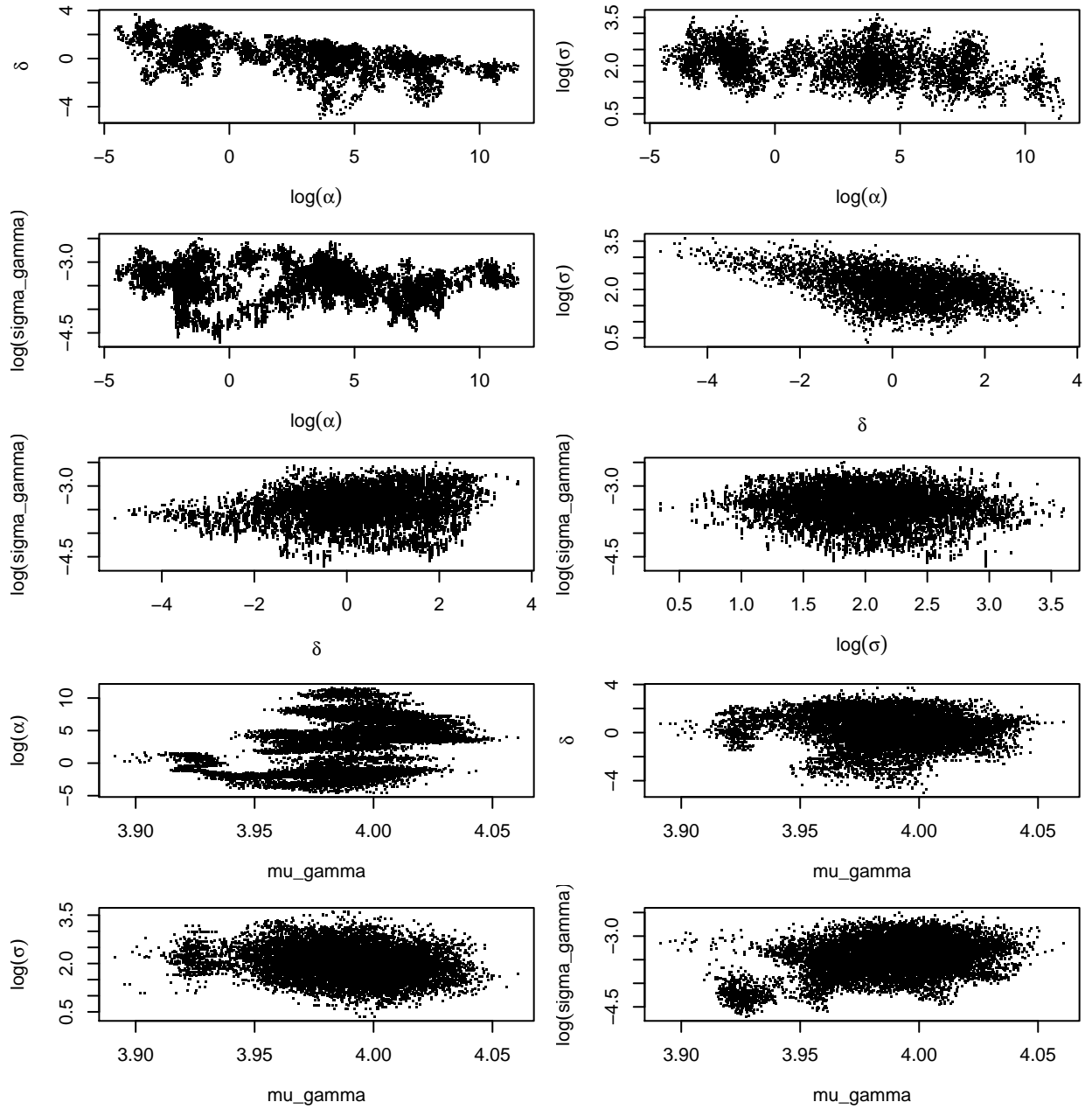


We see that we get quite small values for effective sample size and that the sample size for $\log(\sigma_\gamma)$ is extremely small.

```
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.eff
n.eff
```

```
## [1] 1011.176 1038.662 1132.155 1035.528 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:



We can't see a strong correlation between posterior values, and certainly no nice ellipses we saw in the previous part which allowed us to use a multivariate normal centered on the previous values for our proposal. There does not seem to be an obvious way of improving the metropolis Hastings sampler. Instead we consider numerical integration.