MA40198: Applied Statistical Inference

Candidates: 07604, 07614, 07615

14 December 2018

Carcinogenesis Study on Rats

Abstract

This study seeks to comment on the effectiveness of a carcinopreventive drug following a trial on fifty litters of female rats. Observations were made on how many weeks had passed until a tumour presented in the rats which received treatment as well as a control group which received a placebo.

Under the conditions of this study, there is evidence that the preventative drug had the opposite of the desired effect, causing tumours to present sooner than they otherwise would have.

Introduction

The recorded data includes the variables:

- `litter`, an indicator as to which litter the rat belonged to. Fifty litters were observed, with three rats per litter.
- `rx`, an indicator as to whether treatment was carried out on the rat (`rx` $= 1$). `rx` $= 0$ indicates the rat received the control/placebo only.
- `time`, the follow up time in weeks until the appearance of a tumour.
- `status`, a measure by which to censor the data. `status` $= 1$ indicates the presence of a tumor and `status` $= 0$ indicates that the data was censored due to the death of the rat before the presence of a tumor.

The drug trial was carried out solely on female rats in a controlled environment. One in three of the rats per litter was treated for cancer whilst the other two received a placebo treatment only. Following the treatment, the rats were detained and observed. The time recorded relates to the number of weeks until tumour appearance, or the death of the rat, whichever occured first. This drug trial was carried out with the motivation to discover whether or not the method of cancer treatment was effective in delaying the appearance of cancer in those that are treated.

Using Maximum Likelihood Estimation, this study is comprised of frequentist estimation procedures and Bayesian estimation procedures in order to analyse the observed data.

Methods

Upon the first approach to the MLE problem, the dependent variable `time` described by T_i, has been assumed to follow a Weibull distribution. This seems an appropriate model for the problem due to its ability to describe an 'aging' process within the observed population. That is to say that the occurence of an event, in this case carcinogenesis or death, becomes more likely over time.

The distribution is as below.

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

where

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

It can be seen from the above that the shape parameter in the model takes the form of $\frac{1}{\sigma}$ whilst the scale parameter is proportional to $\exp(\beta_0 + \beta_1 x_i)$. Censored observations, where `status` $= 0$, implies that the rat died before tumour appearance, and are described by the survival function of the Weibull distribution given as follows.

$$S(t|\text{a,b}) = \exp\left(-\left(\frac{t}{b}\right)^a\right), \qquad t > 0$$

The maximum likelihood estimate calculates an estimation of the population parameters such that the likelihood of obtaining the observed data is maximised. The vector of parameters to be estimated is $\boldsymbol{\theta}^T = (\beta_0, \beta_1, \log(\sigma))$.

Given all observations t_i are independent of one another, which we can assume to be true, the likelihood of the whole sample T is the product of the individual likelihoods over all observations t_i, where L is the likelihood. L can be defined as follows.

$$L = L_1 \times L_2 \times ... \times L_N = \prod_{i=1}^{N} L_i$$

Define the maximum likelihood in which the parameters are optimised as follows.

$$P(y_1|\hat{\boldsymbol{\theta}}) \times P(y_2|\hat{\boldsymbol{\theta}}) \times ... \times P(y_N|\hat{\boldsymbol{\theta}}) = \prod_{i=1}^{N} P(y_i|\hat{\boldsymbol{\theta}}) > \prod_{i=1}^{N} P(y_i|\boldsymbol{\theta}) \quad \forall\, \boldsymbol{\theta} \neq \hat{\boldsymbol{\theta}}$$

It is true that the likelihood will always be non-negative and since the natural log is a non-decreasing function, if we transform the likelihood by taking its natural log, the maxima will remain in the same position. When working with log it is computationally easier since for large sample sizes even the maximum likelihood can be below machine precision.

A function to compute the negative log likelihood of the observed times given the parameters $\boldsymbol{\theta}$ is defined in R as below.

```
rats<- read.table("http://people.bath.ac.uk/kai21/ASI/rats_data.txt")
#define negative log likelihood function to minimise
nll.weibull <- function (theta) {
  #initialise variables from dataset
  s <- rats$status
  rx <- rats$rx
  t <- rats$time
  #define shape and scale of Weibull distribution
  k <- 1/exp(theta[3]) #shape
  lambda <- exp(theta[1] + theta[2]*rx) #scale
  #compute negative log likelihood
  loglik <- -sum(log(((k/lambda)*(t/lambda)^(k-1)*exp(-(t/lambda)^k))^s*(exp(-(t/lambda)^k))^(1-s)))
  loglik
}
```

Initially, the function describing the negative log likelihood that will be maximised has been defined as `nll.weibull`. This function utilises the `status` variable to determine the contribution to the likelihood from each observation so that the optimal $\boldsymbol{\theta}$ can be calculated considering the censored and uncensored observations.

In order to compute the log likelihood for the censored observations, the shape and scale parameters of the survival function have been defined below.

$$\text{Shape} = \frac{1}{\sigma}, \qquad \text{Scale} = \exp(\beta_0 + \beta_1 x_i)$$

The scale is altered by `rx`, since we are aiming to estimate the contribution that the treatment made to the observed times. We hereon refer to the shape as k and the scale as λ.

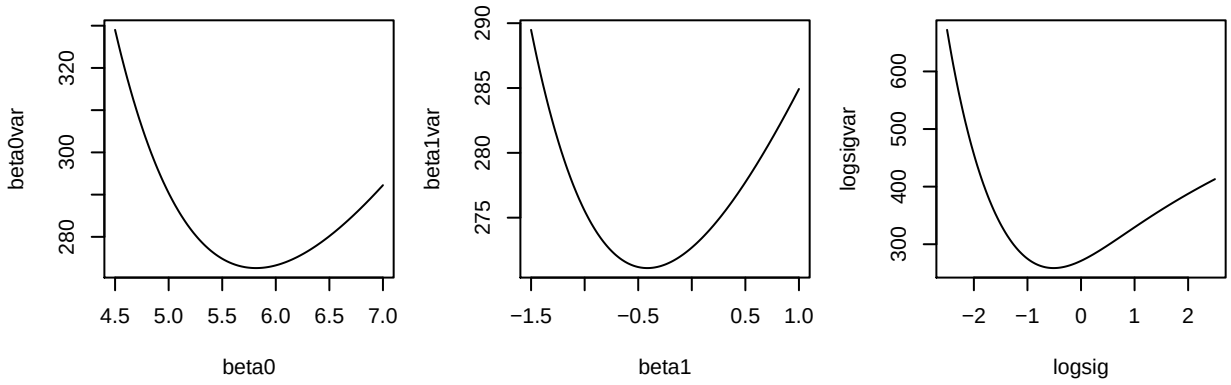
Since the method of finding the MLE has utilised the `optim` function in R, the negative log likelihoods are computed, as `optim` finds the minimum of a given function by default. The likelihood for censored observations has been defined below.

$$(L)_{\texttt{status=0}} = -\sum -\log\left(\exp\left(-\left(\frac{t}{\lambda}\right)^k\right) = \sum\left(\frac{t}{\lambda}\right)^k, \qquad t > 0$$

In order to compute the likelihood of the observations in which the appearance of a tumor occured, the shape and scale parameters define below were passed to the built in R function `dweibull`.

$$\text{shape} = \frac{1}{\sigma}, \qquad \text{scale} = \exp(\beta_0 + \beta_1 x_i)$$

The following plots demonstrate how we chose an appropriate initial point to use when optimising `nll.weibull`. Each parameter within $\boldsymbol{\theta}$ was varied and plotted against the resulting negative log likelihood. The observed minima were chosen as the initial θ.



The minimum for each of the parameters appears to be at approximately $\boldsymbol{\theta} = (5.75, -0.4, -0.5)$ and hence this will be taken as the initial point to proceed with the optimization.

```
theta <- c(5.75,-0.4,-0.5)
weibull.optim <- optim(theta, nll.weibull, method="BFGS", hessian=TRUE)
```

The function `nll.weibull` has been minimised using `optim`, hence the log likelihood has been maximised with respect to the initial values of theta $\boldsymbol{\theta}$ using the BFGS method. The first argument in `optim` provides the initial parameter values $\boldsymbol{\theta}$ from which to start the optimisation. The next argument is the objective function, `nll.weibulll`. The first argument of the function `nll.weibulll` is required to be the vector of parameters which need to be optimised. `optim` has been instructed to return an approximate Hessian matrix at the convergence.

The returned objects are presented below.

```
par
```

```
## [1]  4.9831366 -0.2385118 -1.3326096
```

```
value
```

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

```
counts
```

```
## function gradient
##       39       13
```

```
convergence
```

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

`par` contains the parameter values, the maximum likelihood estimates $\hat{\boldsymbol{\theta}}$. `value` contains the value of the objective function at the minimum and `counts` indicates how many function evaluations were required in order to terminate the optimization. `convergence` when equal to 0 indicates that the `optim` method utilising BFGS converged at a minimum value for the negative log likelihood.

The asymptotic standard error of the parameter estimates is calculated using the Hessian. The inverse of the approximated hessian is the asymptotic variance of the MLE and so the standard errors can be calculated by taking the square root of the diagonal elements of the asymptotic variance. This has been executed in R and shown below.

```
hess <- solve(weibull.optim$hessian)
se <- sqrt(diag(hess))
```

The 95% confidence interval for the paramater β_1 has been calculated with the code below.

```
CI.norand <- c(theta.weibull[2]+qnorm(0.025)*se[2], theta.weibull[2]+qnorm(0.975)*se[2])
```

The 95% asymptotic confidence interval for β_1 is displayed below along with the standard error estimates.

```
## [1] -0.41311362 -0.06391006
```

```
## [1] 0.08332133 0.08908418 0.14387860
```

The correlation matrix has been created using the following code.

```
#estimated correlation matrix of optimal theta
corrmat <- diag(1/se)%*%hess%*%(diag(1/se))
```

```
corrmat
```

```
##             [,1]       [,2]       [,3]
## [1,]  1.0000000 -0.7324901  0.6873689
## [2,] -0.7324901  1.0000000 -0.3478359
## [3,]  0.6873689 -0.3478359  1.0000000
```

Looking at the correlation matrix we can see that β_1 has quite a strong negative correlation of approximately -0.7325 with β_0 and β_1, and $\log(\sigma)$ are moderately correlated with one another with a value of -0.348. The optimal parameter estimate for β_1 has been approximated at approximately -0.2384 and the 95% asymptotic confidence interval for this value is ranging over the values $(-0.4131, -0.0640)$. This interval seems relatively wide considering the context of the study. For example, the absolute value of this range can be approximated at 0.3496 which is greater than the magnitude of the parameter estimate itself. Any estimate for β_1 falling within this range has a 95% certainty that it fits the true population.

The data will now be analysed on the the probability model for T_i takes on a log-logistic distribution as seen below with the same shape and scale parameters as in the Weibull model.

$$T_i \sim \text{log-logistic}(\text{shape} = \frac{1}{\sigma}, \text{scale} = \exp(\eta_i)), \qquad \eta_i = \beta_0 + \beta_1 x_i$$

Whilst the survival function of a log-logistic distribution can be expressed as.

$$S(t|\text{a,b}) = \exp\left(-\log\left(1 + \left(\frac{t}{b}\right)^a\right)\right), \qquad t > 0$$

Hence, the negative log likelihood of the censored observations can be written as below.

$$-\log(L_{\texttt{status}=0}) = -\sum \log\left(\exp\left(-\log\left(1+\left(\frac{t}{\lambda}\right)^{k}\right)\right) = \sum \log\left(1+\left(\frac{t}{\lambda}\right)^{k}\right), \quad t>0$$

In equal fashion to the Weibull distribution, the collective likelihood of both censored and uncensored observations will be written as the sum of $-\log(L_{\texttt{ststus}=0})$ and $-log(L_{\texttt{status}=1})$. The function that will be minimised has been defined in the R code below.

```
#define negative log likelihood function to minimise
nll.llogistic <- function (theta) {
  #initialise variables from dataset
  t <- rats$time
  s <- rats$status
  rx <- rats$rx
  #define shape and scale of log-logistic distribution
  k <- 1/exp(theta[3]) #shape
  lambda <- exp(theta[1] + theta[2]*rx) #scale
  #compute negative log likelihood
  loglik <- -sum(log(((k/lambda)*(t/lambda)^(k-1)/(1+(t/lambda)^k)^2)^s*(1/(1+(t/lambda)^k))^(1-s)))
  loglik
}
```

The function, `nll.logistic` that has been defined above will be passed to `optim` using an initial vector of parameters estimated in the same fashion as previously within the report in which we examine the change in likelihood caused by varying each parameter.

The function will be optimized with respect to the chosen initial point $\boldsymbol{\theta}$. The R code constructed in order to achieve this is below.

```
theta <- c(5.75,-0.5,-0.5) #estimated starting theta
nll.optim <- optim(theta, nll.llogistic, hessian=TRUE, method="BFGS")
```

The objects returned by `nll.optim` are:

`par`

```
## [1]  4.9165334 -0.2295086 -1.4102831
```

`value`

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

`counts`

```
## function gradient
##       35       12
```

`convergence`

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

It can be seen that from the `counts` output, that 35 function evaluations and 12 gradient evalutions were required in order to converge to the optimal solution in this case.

It is now left to estimate the asymptotoic standard errors of each parameter estimate. This along with a 95% confidence interval is estimated using the R code below.

```
hess <- nll.optim$hessian #name the nll.optim hessian
#to find the standard error, take the sqrt of the diag of the inverse hessian
se <- sqrt(diag(solve(hess)))
#95% CI = parameter estimate +- ~1.96*stderr
CI <- c(theta.llog[2] + qnorm(0.025)*se[2], theta.llog[2] + qnorm(0.975)*se[2])
```

The asymptotic standard error of each of the parameter estimates $\hat{\boldsymbol{\theta}}$, along with the 95% asymptotic confidence interval for the treatment effect β_1.

```
## [1] 0.08078076 0.09560751 0.14142432
```

```
## [1] -0.41689587 -0.04212134
```

It can be seen that the asymptotic standard error of β_1 has been approximated at a value of 0.0956. Whilst the 95% confidence interval for β_1 is computed as $(-0.4169, -0.0421)$. Comparatively to the MLE of the Weibull distribution, this confidence interval has increased slightly in magnitude. Hence there is a greater range of values that could be taken to be the parameter estimate, such that the true mean of the population is with 95% certainty.

Proceeding onward, the Weibull distribution was again considered, but with added random effect describing the litter, $\boldsymbol{b}$. So it can be seen that the scale of the Weibull distribution is then altered to the exponent of η_i. Where η_i is now as follows:

$$\eta_i = \beta_0 + \beta_1 x_i + b_{litter(i)}$$

In order to obtain the likelihood of such a distribution given the data and the added random effect, $\boldsymbol{b}$, the integral must be found as well as the joint density. In order to surpass the intractable integral, we use the Laplace approximation to obtain the likelihood.

So that the Laplace approximation can be utilised to obtain the likelihood, a function `lfyb` has been created. This evaluates the negative joint log density of the observations $\boldsymbol{y}$ and the added random effect, *litter* $\boldsymbol{b}$ given the initial vector of parameters $\boldsymbol{\theta}$. It then follows that in order to use the Laplace approximation for the likelihood, the minimum of the negative log joint density of the random effect $\boldsymbol{b}$ and the data $\boldsymbol{y}$ as well as the relevant Hessian must be obtained.

Hence, it is necessary to create the function `lfyb` such that the arguments of the function will be $\boldsymbol{\theta}$, $\boldsymbol{y}$, $\boldsymbol{b}$, X and Z. The matrices X and Z allow for the relevant variables to be selected for any given observation. The following R code is a means of constructing these arguments.

```
#theta = [beta0, beta1, log(sigma), log(sigmab)]
X <- model.matrix(~ rx + status, rats) #define X matrix
rats$litter <- factor(rats$litter)
Z <- model.matrix(~ litter - 1, rats) #define Z matrix
b <- rnorm(50,0,0.1) #set arbitrary b
y <- rats$time #define y vector
theta <- c(4.9831505, -0.2384417, -1.3324342, 0) #using optimal paramater estimates as initial
#from Q1, & small log(sig.b)
```

These arguments are then used to construct the function `lfyb` as such:

```
lfyb <- function (b, y, theta, X, Z) {
  #function to compute the joint log density of y and b, which is the
  #(conditional log likelihood of y given b) + (likelihood of b)
  #first compute conditional density of y given b
  s <- X[,3] #status
  beta <- c(theta[1:2],0) #define beta vector with a 0 in the last position since status is not relevan
  eta <- as.numeric(X%*%beta + Z%*%b) #define eta vector
  k <- 1/exp(theta[3]) #shape
  lambda <- exp(eta) #scale vector
  #log conditional likelihood of y given b
  nll1 <- k*log(lambda) - log(k) - (k-1)*log(y) + (y/lambda)^k #scale = 1
  nll0 <- (y/lambda)^k #scale = 0
  lfy_b <- sum(s*nll1 + (1-s)*nll0)
  #negative log likelihood of b
  sig.b <- exp(theta[4])
```

```
  #lfb <- -sum(dnorm(b, 0, sig.b, log=TRUE))
  lfb <- dnorm(b, 0, sig.b)
  lfb <- log(lfb)
  lfb[lfb==-Inf] <- -10
  lfb <- -sum(lfb)
  #compute and output negative joint log density
  lf <- lfy_b + lfb
  lf
}
```

Following this, the gradient has been computed symbolically using the following R code in order to make optimisation more efficcient in fifty dimensions.

```
#compute the derivative for status = 1
expr1 <- expression(k*(B0 + B1*rx + b.aux) + (y/exp(B0 + B1*rx + b.aux))^k)
nll.aux1 <- deriv(expr1, c("b.aux"), function.arg=c("b.aux", "B0", "B1", "k", "rx", "y"))

#status = 0
expr0 <- expression((y/exp(B0 + B1*rx + b.aux))^k)
nll.aux0 <- deriv(expr0, c("b.aux"), function.arg=c("b.aux", "B0", "B1", "k", "rx", "y"))

#numerically compute the gradient of lfyb
grad <- function (b, y, theta, X, Z) {
  #define variables
  rx <- X[,2]
  s <- X[,3]
  B0 <- theta[1]
  B1 <- theta[2]
  k <- 1/exp(theta[3])
  b.aux <- Z%*%b
  #gradient if status = 1
  b.grad1 <- nll.aux1(b.aux, B0, B1, k, rx, y)
  g1 <- attr(b.grad1, "gradient")
  g1 <- as.numeric(g1)
  #gradient if status = 0
  b.grad0 <- nll.aux0(b.aux, B0, B1, k, rx, y)
  g0 <- attr(b.grad0, "gradient")
  g0 <- as.numeric(g0)
  #select contribution to gradient based on status
  g <- s*g1 +(1-s)*g0
  #gradient of negative log likelihood of b
  b.grad <- b/exp(2*theta[4])
  m <- matrix(g, nrow=3, ncol=50)
  g <- colSums(m) + b.grad #add contributions from b terms
  g
}
```

It then follows to create a function `lal` for given parameters $\boldsymbol{\theta}$. We can optimize $\boldsymbol{b}$ within `lal`. This function takes the optimal values for $\boldsymbol{b}$, $\hat{\boldsymbol{b}}$ to then find the Laplace approximation for $\hat{\boldsymbol{\theta}}$, given $\hat{\boldsymbol{b}}$. This has been done using the R code below.

```
lal <- function (theta, y, X, Z) {
  #compute the negative log likelihood of theta using the laplace approximation
  theta[theta<(-3)] <- -3
  theta[theta > 7] <- 7
```

```
  b <- rep(0,50)
  lfyb.opt <- optim(b, fn=lfyb, gr=grad, theta=theta, y=y, X=X, Z=Z, method="BFGS", hessian=TRUE)
  b.opt <- lfyb.opt$par
  hess <- lfyb.opt$hessian
  lapprox <- -(length(b.opt)/2)*log(2*pi) + (1/2)*log(det(hess)) + lfyb(b=b.opt, y=y, theta=theta, X=X,
  lapprox
}
 lal.opt <- optim(theta, lal, y=y, X=X, Z=Z, method="BFGS", hessian=TRUE)
```

The optimal vector of parameters, $\hat{\boldsymbol{\theta}}$ is

```
## [1]  5.0085840 -0.2308727 -1.3780455 -1.6596795
```

The Weibull distribution adopted previously to model the effects of the added random effects, will be analysed again but this time following a Bayesian estimation procedure. The proposal distributions $q(\boldsymbol{\theta}_j|\boldsymbol{\theta}_{j-1})$ which were used to initiate the Metropolis Hastings method was chosen to be the marginal negative log-likelihood. The starting parameter $\boldsymbol{\theta}_0$ was taken to be the previously computed optimal values $\hat{\boldsymbol{\theta}}$ in the early stages of the analysis along with a small parameter estimate for $\log(\sigma_b)$.

$$\boldsymbol{\theta}_0 = (4.9831505, -0.2384417, -1.3324342, 0)$$

The R code used to define the initial value paramters can be seen below.

```
#we set the seed for random generators for reproducibility
set.seed(4)

X <- model.matrix(~ rx + status, rats) #define X matrix
rats$litter <- factor(rats$litter)
Z <- model.matrix(~ litter - 1, rats) #define Z matrix
b <- rnorm(50,0,0.1) #set arbitrary random b
y <- rats$time #define y vector
theta <- c(4.9831505, -0.2384417, -1.3324342, 0) #using optimal paramater estimates from Q1, & small lo
```

The defined parameters are passed as arguments to the log posterior function below. $\log(\sigma_b)$ follows an exponential prior distribution with rate 5 and the remaining parameters have uniform prior distributions. The function `lfyb` that was previously computed has been made positive and added to the prior for $\log(\sigma_b)$.

```
log.posterior <- function (b, theta, times = y, X.mat=X, Z.mat=Z) {
  #log likelihood
  prior <- dexp(x=exp(theta[4]), rate=5, log=TRUE) #define log(sigb) to follow a exponentional prior di
  prior[prior==-Inf] <- -10
  l.lik <- -lfyb(b, times, theta, X.mat, Z.mat) + prior #add the absolute values of the log likelihood
  l.lik
}
```

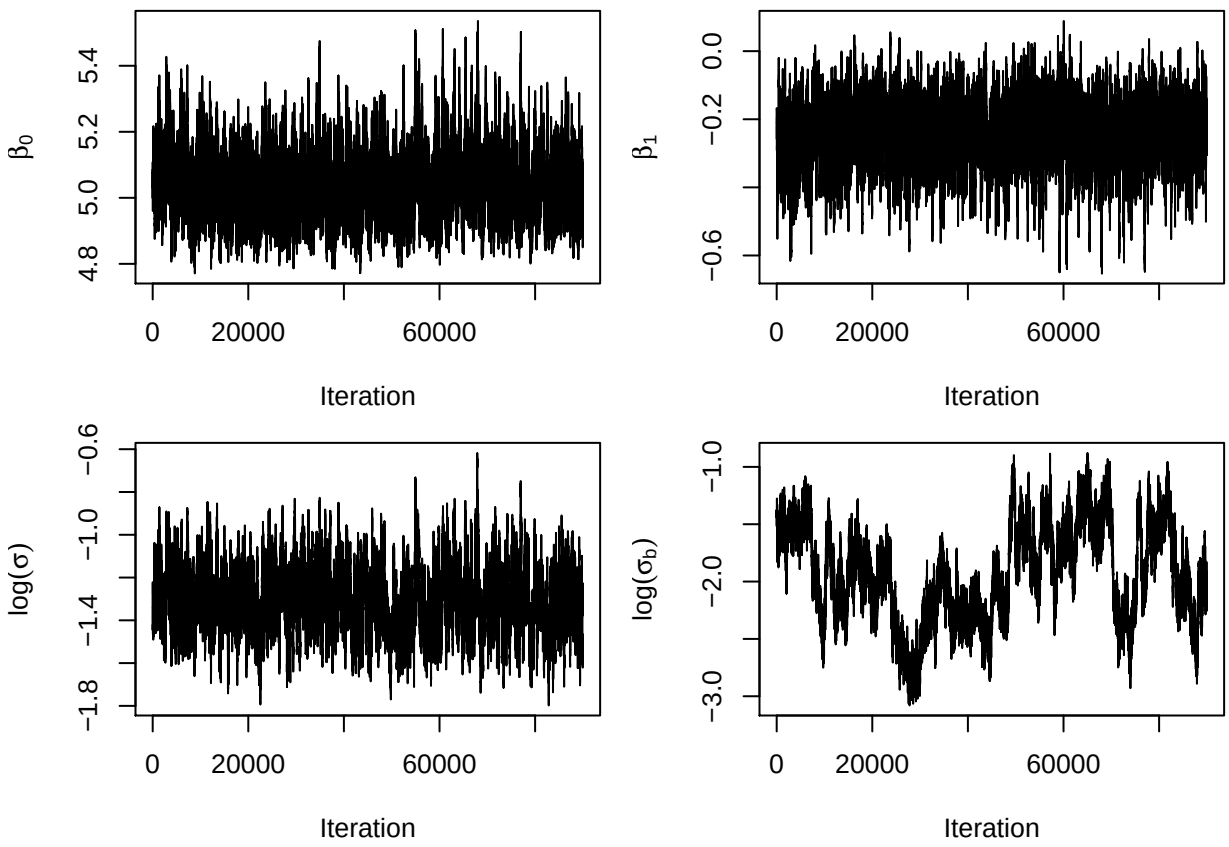
The metropolis Hastings algorithm is implemented with the code below. The proposal distributions used

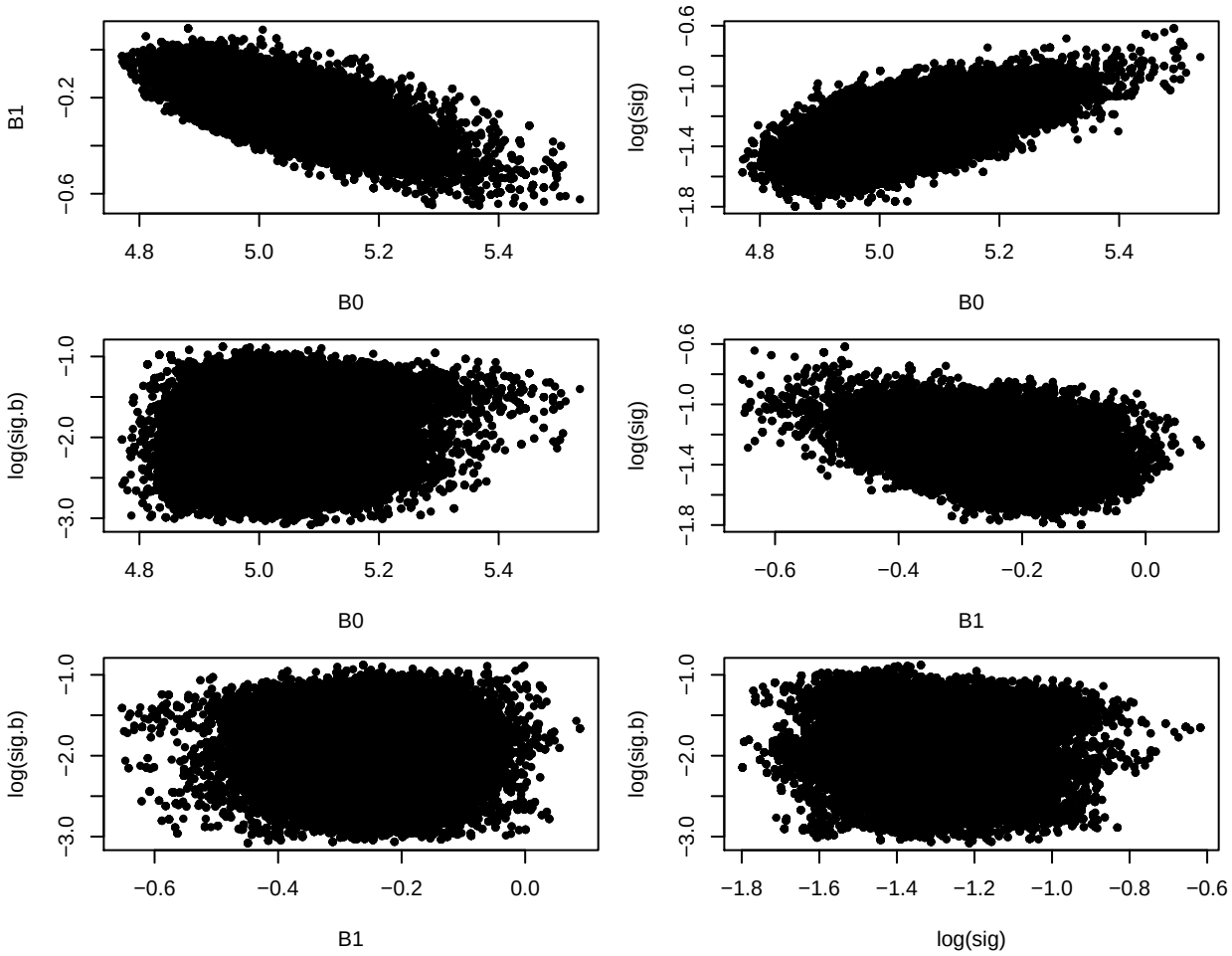
```
n.rep <- 100000
sigma.prop <- c(0.15, 0.055, 0.055, 0.1) #tuning possible; (0.15, 0.055, 0.055, 0.1)
#looks to be very good in terms of acceptance rate
MH <- function (theta, sigma.prop, n.rep) {
  theta.vals <- matrix(0, n.rep, 4) #matrix to save generated values
  theta.vals[1,] <- theta
  b <- rep(0, 50)
  b.vals <- matrix(0, n.rep, 50)
  b.vals[1,] <- b
  lp0 <- log.posterior(b, theta.vals[1,])
```

```
  accept.th <- 0
  accept.b <- 0
  for (i in 2:n.rep) {
    #update theta
    theta <- theta+rnorm(4, 0, sigma.prop)
    lp1 = log.posterior(b, theta)#evaluate log posterior
    if (runif(1) < exp(lp1 - lp0)){#accept step with probability alpha
      accept.th <- accept.th+1
      lp0 <- lp1
    }else{
      theta <- theta.vals[i-1,]
    }
    #update random effects
    b <- b + rnorm(50)*0.045 #after tuning, 0.045 produces an acceptance rate around 0.23
    lp1 <- log.posterior(b, theta)
    if (runif(1) < exp(lp1 - lp0)){
      accept.b <- accept.b+1
      lp0 <- lp1
    }else{
      b <- b.vals[i-1,]
    }
    theta.vals[i,] <- theta
    b.vals[i,] <- b
  }
  accept.rate <- c(accept.th/n.rep, accept.b/n.rep)
  list(theta=theta.vals, accept.rate=accept.rate)
}
```

```
#initialize theta to an arbitrary value; run Metropolis Hastings
theta0 <- c(1,0,0.1,0.2)
mh <- MH(theta0, sigma.prop, n.rep)
theta <- mh$theta
accept.rate <- mh$accept.rate
```

```
par(mfrow=c(2,2),mar=c(4,4,1,1))
lower <- n.rep/10+1
yaxis <- c(expression(beta[0]), expression(beta[1]), expression(log(sigma)), expression(log(sigma[b])))
for (i in 1:4){
  plot(mh$theta[lower:n.rep,i], type="l", ylab=yaxis[i], xlab="Iteration")
}
```


β0
4.8
5.0
5.2
5.4
0
20000
60000
Iteration
β1
−0.6
−0.2
0.0
0
20000
60000
Iteration
log(σ)
−1.8
−1.4
−1.0
−0.6
0
20000
60000
Iteration
log(σb)
−3.0
−2.0
−1.0
0
20000
60000
Iteration



Need to talk about the plots.

Fatigue of materials

Absract

A number of fatigue tests have been carried out on materials which are subject to cyclic loading.

This study has been set up as a means of predicting the fatigue, as to aid the use of fatigue materials within the practice of mechanical and structural engineering.

Introduction

The fatigue characteristics are established through 26 fatigue tests which are performed on the small flat plates of the loading materials called coupons. The following elements of the study were recorded.

- `Cycles` the number of cycles, N_i until failure in coupon i.
- `Stress levels` the controlled levels of stress, s_i have been recorded as force per unit area (in Mega Pascals, MPa).

The stress levels s_i are controlled throughout the series of tests, whilst the number of cycles to failure N_i exhibits a random behaviour due to inherent microstructural inhomogeneity in the material properties as

well as due to uncontrolled differences in test conditions. If a test is terminated before the coupon fails, then the coupon is market as a 'runout'. In terms of likelihoods, the runouts can be interpreted as censored observations.

Methods

The number of cycles has been modelled using the following probability model.

$$N_i = \alpha(s_i - \gamma)^{\delta}\epsilon_i, \qquad \text{where} \quad s_i > \gamma$$

and ϵ_i is a random error such that,

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

The constants $\alpha > 0$, $\delta \in \mathbb{R}$, $\delta > 0$ and $\sigma > 0$ are unknown parameters. Empirical results suggest coupons tested below the stress level γ will never fail. The unknown parameter γ is therefore called the **fatigue limit**.

Primarily, γ was taken to be 50 so that the runouts will be ommitted. The maximum likelihood estimate of $\boldsymbol{\theta} = (\log(\alpha), \delta, \log(\sigma))^T$, was found using `optim`. The R code constructed to compute this is displayed.

```
fatigue<- read.table("http://people.bath.ac.uk/kai21/ASI/fatigue.txt")
#need to show the Ni is a weibull dist with shape epsilon and scale alpha(s-gama)^delta,
#then can compute likelihood as in Q1.
#Question 2.1
gama <- 50 #arbitrary gamma
theta <- c(1,1,1) #arbitrary initial theta
s <- fatigue$s #stress
N <- fatigue$N
nll.weibull <- function (theta, gama, s, N) {
  k <- 1/exp(theta[3]) #shape
  lambda <- exp(theta[1])*(s - gama)^theta[2] #scale
  ro <- fatigue$ro
  #compute the negative log likelihood
  nll0 <- -sum(dweibull(N, shape=k, scale=lambda, log=TRUE)^(1-ro))
  nll1 <- sum(((N/lambda)^k)^ro)
  nll <- nll0 + nll1
  nll
}
maxlik <- optim(theta, nll.weibull, gama=gama, s=s, N=N, method="BFGS", hessian=TRUE)
```

The returned objects can be seen below.

```
## $par
## [1] 23.1269672 -3.1577757 -0.8702683
##
## $value
## [1] 265.5036
##
## $counts
## function gradient
##       73       31
##
## $convergence
## [1] 0
##
```

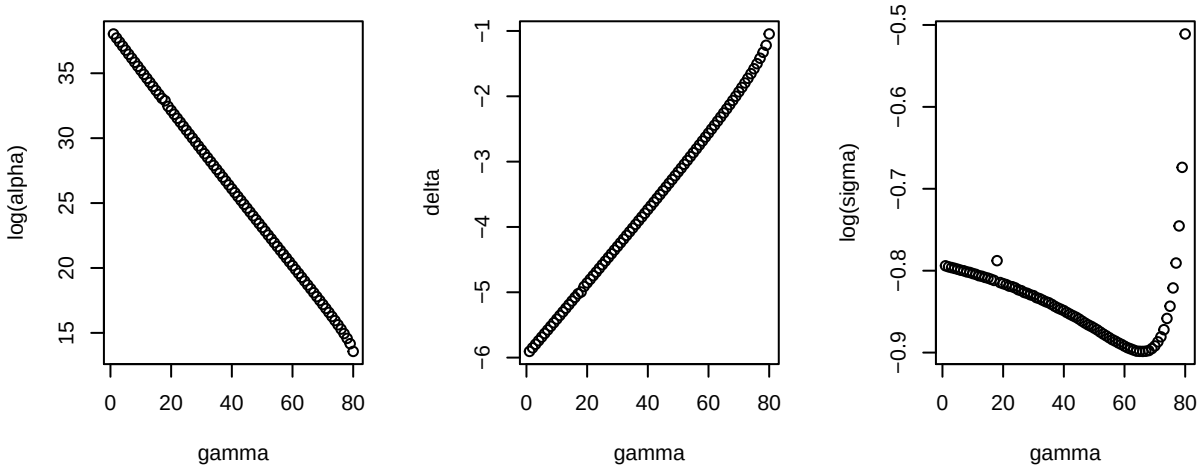
```
## $message
## NULL
##
## $hessian
##           [,1]       [,2]     [,3]
## [1,] 125.41197  495.24993 10.64051
## [2,] 495.24993 1977.14997 46.54281
## [3,]  10.64051   46.54281 34.10709
```

The optimised vector of parameters $\hat{\boldsymbol{\theta}} = (13.5727280, -1.0460720, -0.5109903)$. The asymptotic 95% confidence intervals of the parameters are displayed in columns 1, 2 and 3 respectively below.

```
##          [,1]      [,2]       [,3]
## [1,] 21.42556 -3.587532 -1.2154052
## [2,] 24.82837 -2.728019 -0.5251313
```

γ was then varied in order to identify the impact it has on the sensitivity of the results. The code that has been displayed below and following this, plots were made in orer to observe the trend.

```
#How variations in gama affects the theta
f.vals <- rep(0,80)
f.par <- matrix(0,nrow=3, ncol=80)
for (i in 1:80) {
  gama <- i
  f.vals[i] <- nll.weibull(theta, gama, s, N)
  maxlik <- optim(theta, nll.weibull, gama=gama, s=s, N=N, hessian=TRUE)
  f.par[,i] <- maxlik$par
}
```



There seems to be a downward linear trend amongst $\log(\alpha)$ and γ. As the values for γ increase from 0 to 80, $\log(\alpha)$ decreases from around 40 for $\gamma = 0$. There is an apparent upward linear trend amongst δ for increasing γ. It varies from around -6 to around -1. As γ increases, δ appears to relatively steadily decline from around -0.8 to -0.9 and then randomly seems to increase exponentially. It appears that changes in γ impact δ quite substantially. This parameter is sensitive to change in γ.

γ has been estimated amongst the vector of unknowns $\boldsymbol{\theta}_* = (\log(\alpha), \delta, \log(\sigma), \gamma)$. This has been executed using the *R* code below.

```r
#We can estimate gama by including it within theta and passing it to optim
#redefine nll.weibull with theta[4] = gama
gama <- 50 #reset gama
theta.opt <- c(theta.opt, gama)
nll.weibull2 <- function (theta, s, N) {
  k <- 1/exp(theta[3]) #shape
  lambda <- exp(theta[1])*(s - theta[4])^theta[2] #scale
  ro <- fatigue$ro
  #compute the negative log likelihood
  nll0 <- -sum(dweibull(N, shape=k, scale=lambda, log=TRUE)^(1-ro))
  nll1 <- sum(((N/lambda)^k)^ro)
  nll <- nll0 + nll1
  nll
}
maxlik2 <- optim(theta.opt, nll.weibull2, s=s, N=N, hessian=TRUE)
theta.opt2 <- maxlik2$par
gama.opt <- theta.opt2[4] #gama looks like it should be around 66
```

It has been done in such a way that γ has been passed to the vector of parameters which are modelled using a Weibull distribution. The new vector of parameters $\boldsymbol{\theta}_*$ which ultimately will be passed to `optim` to compute the MLE. The estimates of the new vector of parameters is displayed below.

```
## [1] 18.2544956 -2.1624241 -0.8984666 66.4694806
```

Hence, with γ as an added parameter it has been estimated to be around 66.

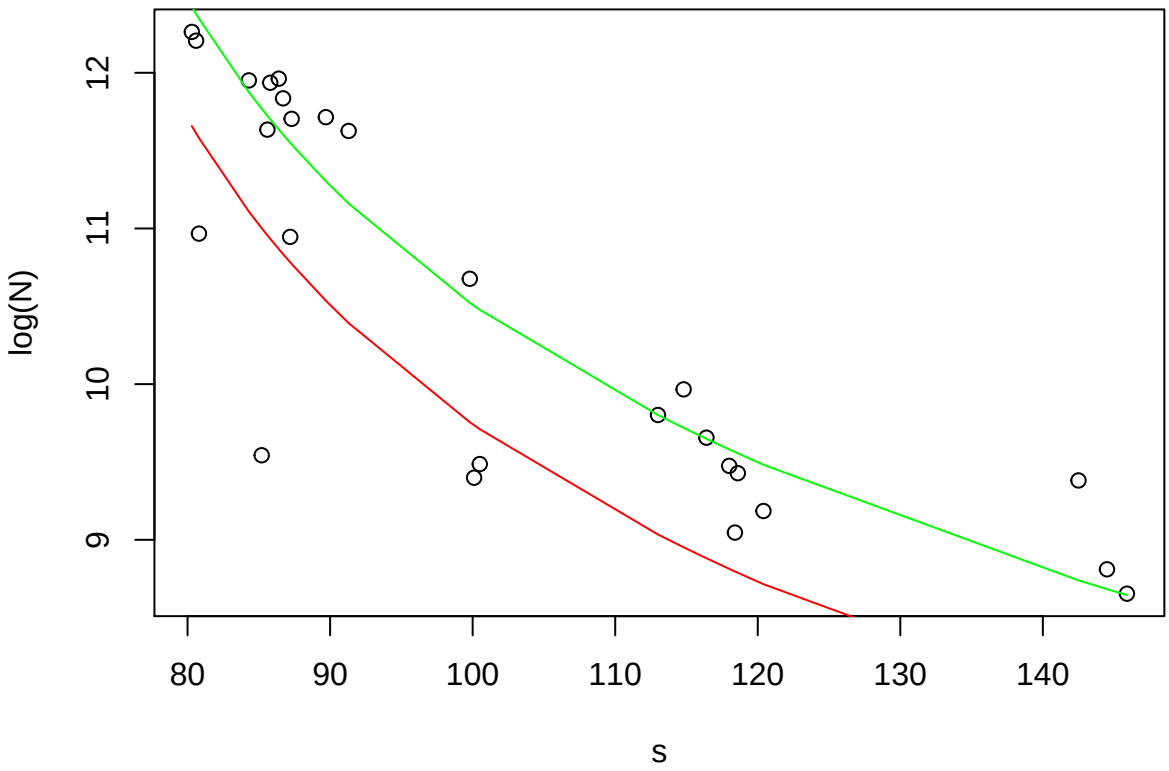
We proceed to focus on the lower quantiles of the fatigue as a function of stress as this is an area of great interest within engineering. The lower 10% quantile of N is considered using the following probability model.

$$N_{0.1} = \alpha(s_i - \gamma)^{\delta} z_{0.1}^{\sigma}$$

Where $z_{0.1}$ is the lower quantile of a Weibull distribution with unit shape and scale. This will be such that it is am exponential distribution with unit rate. The following R code displays the construction of the 10% quantile.

```r
#function to compute quantile of N
N.quantile <- function (theta, s, quantile) {
  ZQ <- qweibull(quantile, shape=1, scale=1)
  NQ <- exp(theta[1])*(s-theta[4])^theta[2]
  Q <- NQ*ZQ^exp(theta[3])
  Q
}
```

The plot below displays the log transformed data points along with interpolated 10% and 50% quantiles of N.



It can be seen that the estimated MLE parameters calculated fits the observed data quite well, however there are some obvious outliers that fall below the 10% quantile of the curve.

Previously, we assumed the fatigue limit of all the coupons to be the same. We can modify our model to assume that the fatigue limit of the coupons are different by introducing random effects for each of the coupons. Let the set of random effects be Γ. We require that all $\gamma_i < s_i$. The probability density function of each Γ_i is Weibull(shape $= \frac{1}{\sigma_\gamma}$, scale $= \exp(\mu_\gamma)$). The conditional probability of the number of cycles to failure given the fatigue limit of a coupon is $N_i|\Gamma_i = \gamma_i < s_i \sim$ Weibull(shape $= \frac{1}{\sigma}$, scale $= \alpha(s_i - \gamma_i)^\delta$) If we let $\boldsymbol{b}$ be the vector of random effects we can define the vector of unknown parameters as $\boldsymbol{\theta}^T = (log(\alpha), \delta, log(\sigma), \mu_\gamma, log(\sigma_\gamma))$. To implement a Metropolis Hastings algorithm, we assume uniform improper prior distributions for all the parameters except for σ_γ, which has an exponential prior distribution with rate 5. We also assume that the parameters are independent of each other. We can then make use of the same Metropolis-Hastings algorithm as in the previous study to simulate from the posterior of $\boldsymbol{\theta}$. The log posterior is as follows

```
log.post <- function (theta, gama, s.=s, ro.=ro, N.=N) {
  #log density of N|gama
  k <- 1/exp(theta[3]) #shape
  lambda <- exp(theta[1])*(s. - gama)^theta[2] #scale
  ll0 <- sum(dweibull(N., shape=k, scale=lambda, log=TRUE)*(1-ro.))
  ll1 <- sum((-(N./lambda)^k)*ro.)
  ll <- ll0 + ll1
  if (is.nan(ll) || ll == -Inf) {
    ll <- -1e3
  }
```

```
  #log density of gama
  ll.gama <- sum(dweibull(gama, shape = 1/exp(theta[5]), scale = exp(theta[4]), log=TRUE))
  if (is.nan(ll.gama) || ll.gama == -Inf) {
    ll.gama <- -1e3
  }
  #log density of prior of sig.gama
  ll.prior <- dexp(exp(theta[5]), rate=5, log=TRUE)
  if (ll.prior == -Inf) {
    ll.prior <- -1e3
  }
  #sum of log densities
  lp <- ll + ll.gama + ll.prior
  lp
}
```

We use the log posterior to decide whether a proposed step should be accepted or not inside the Metropolis-Hastings algorithm. We then implement the algorithm in an R function as below.

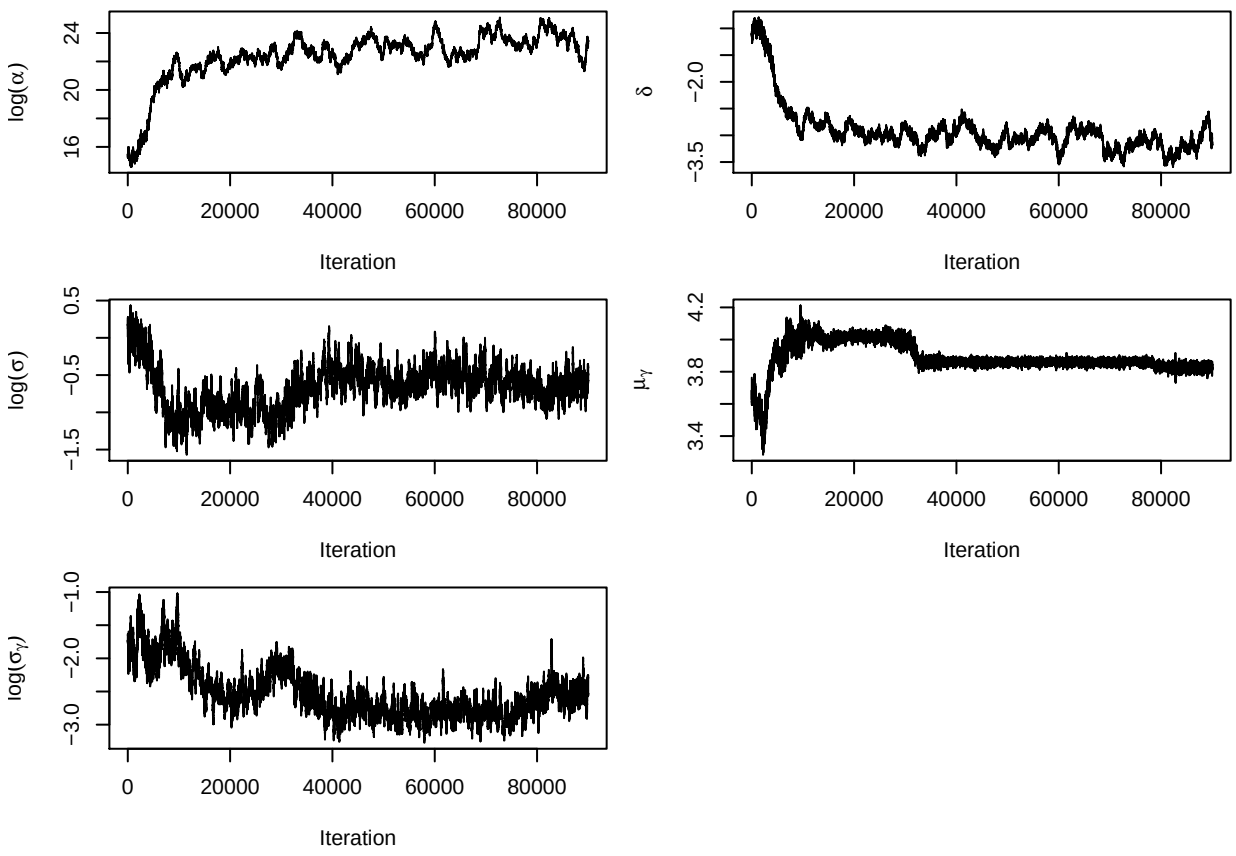
```
MH <- function (theta, sigma.prop, n.rep, s.=s, ro.=ro, N.=N) {
  theta.vals <- matrix(0, n.rep, 5) #matrix to save generated values
  theta.vals[1,] <- theta
  b <- runif(26, min=0.01, max=s.)
  b.vals <- matrix(0, n.rep, 26)
  b.vals[1,] <- b
  lp0 <- log.post(theta.vals[1,], b)
  accept.th <- 0
  accept.b <- 0

  for (i in 2:n.rep) {
    #update theta
    theta <- theta + rnorm(5, 0, sigma.prop[1:5])
    lp1 <- log.post(theta, b)
    if (runif(1) < exp(lp1 - lp0)){
      accept.th <- accept.th+1
      lp0 <- lp1
    }else{
      theta <- theta.vals[i-1,]
    }
    #update random effects
    b.step <- sigma.prop[6]
    b <- b + runif(26, -s., s.)*b.step
    #we restrict b to staying between 0 and the corresponding element in s
    while (length(b[b<0 | b>s.]) > 0) {
      b[b<0 | b>s.] <- b.vals[i-1,which(b<0 | b>s.)] + runif(length(b[b<0 | b>s.]), -s., s.)*b.step
    }
    #we evaluate the log posterior at the new b and accept or reject the step
    lp1 <- log.post(theta, b)
    if (runif(1) < exp(lp1 - lp0)){
      accept.b <- accept.b+1
      lp0 <- lp1
    }else{
      b <- b.vals[i-1,]
    }
    theta.vals[i,] <- theta
```

```
    b.vals[i,] <- b
  }
  accept.rate <- c(accept.th/n.rep, accept.b/n.rep)
  list(theta=theta.vals, accept.rate=accept.rate, gama=b.vals)
}
```

This returns every θ and random effect Γ visited as well as the acceptance rate for steps in the algorithm. To call the function we load the data, fix an initial guess for θ, the number of repetitions and the spread of the proposed steps. An example is below.

```
fatigue<- read.table("http://people.bath.ac.uk/kai21/ASI/fatigue.txt")
#set seed for reproducibility
set.seed(7)
s <- fatigue$s
N <- fatigue$N
ro <- fatigue$ro
theta <- c(5, -2, 0, 4, -1.5)
sigma.prop <- c(0.055, 0.05, 0.05, 0.05, 0.05, 0.1)#we achieve an optimal acceptance rate using these p
n.rep <- 100000
mh <- MH(theta, sigma.prop, n.rep)
```

```
par(mfrow=c(3,2),mar=c(4,4,1,1))
yaxis <- c(expression(log(alpha)), expression(delta), expression(log(sigma)), expression(mu[gamma]), ex
for (i in 1:5){
  plot(mh$theta[lower:n.rep,i], type="l", ylab=yaxis[i], xlab="Iteration")
}
```



Given the actual marginal density of each n_i, we can redefine our log posterior function to include the marginal of N. The marginal of n_i is

$$f(n_i) = \int_0^{s_i} f(n_i|\gamma_i) f(\gamma_i) d\gamma_i$$

Then the marginal of N is the product of all $f(n_i)$ since the failure times are assumed to be independent. To implement this in R, we define the function to be integrated for each i as below

```
f <- function (g, i, theta) {
  fng <- dweibull(N[i], shape = 1/exp(theta[3]), scale = exp(theta[1])*(s[i]-g)^theta[2])
  fg <- dweibull(g, shape = 1/exp(theta[5]), scale = exp(theta[4]))
  f <- fng*fg
  f
}
```

Since we need the natural logarithm of the product of these integrals, we can define a loop to compute the integral for each i, and sum up the logarithms as in the code below

```
I <- rep(0,26)
for (i in 1:length(s)){
  G <- integrate(f, lower=0, upper=s[i], i=i, theta=theta)
  I[i] <- G$value
}
I <- sum(log(I))
```

This loop can be used in the log posterior function. The following is the same as in the previous part except for the addition of the term defined by the integral. Since the marginal is the denominator in the expression for the posterior distribution, the value stored in the variable I has to be subtracted in the function.

```
log.post <- function (theta, gama, s.=s, ro.=ro, N.=N) {
  #log density of N|gama
  k <- 1/exp(theta[3]) #shape
  lambda <- exp(theta[1])*(s. - gama)^theta[2] #scale
  ll0 <- sum(dweibull(N., shape=k, scale=lambda, log=TRUE)*(1-ro.))
  ll1 <- sum((-(N./lambda)^k)*ro.)
  ll <- ll0 + ll1
  if (is.nan(ll) || ll == -Inf) {
    ll <- -1e3
  }
  #log density of gama
  ll.gama <- sum(dweibull(gama, shape = 1/exp(theta[5]), scale = exp(theta[4]), log=TRUE))
  if (is.nan(ll.gama) || ll.gama == -Inf) {
    ll.gama <- -1e3
  }
  #log density of prior of sig.gama
  ll.prior <- dexp(exp(theta[5]), rate=5, log=TRUE)
  if (ll.prior == -Inf) {
    ll.prior <- -1e3
  }
  #marginal of N
  I <- rep(0,26)
  for (i in 1:length(s.)){
    G <- integrate(f, lower=0, upper=s.[i], i=i, theta=theta)
    I[i] <- G$value
  }
  I <- sum(log(I))
```

```
  #sum of log densities
  lp <- ll + ll.gama + ll.prior - I
  lp
}
```

Since the rest of the model is unchanged, we can use the rest of the implementation of the Metropolis Hastings algorithm as before.

When we run both programs, we see that the value of the parameters is in the same range for both methods, as expected. However, when using the marginal density of each n_i, the random walk seems to be more sticky, meaning the autocorrelation of the parameters is quite high, which necessitates a larger sample size to obtain good results. This can become a problem especially since the R function `integrate` takes a long time to run. In addition, the parameters in θ seem more correlated than they were without using the marginal likelihood in the posterior density. Plots of the random walk are below (again discarding the first 20% of the random walk)

```
theta <- c(5, -2, 1, 4, -1.5)
sigma.prop <- c(0.11, 0.11, 0.11, 0.11, 0.11, 0.11)
n.rep <- 10000
mh <- MH(theta, sigma.prop, n.rep)
lower <- n.rep/5+1
```