HW18

Zach White

11/22/2016

Exercise 3

```r
# Create a data list with inputs for JAGS

n = nrow(stackloss)
## scale X such that X^TX has ones on the diagonal;
## scale divides by the standard deviation so we need
## to divide by the sqrt(n-1)
scaled.X = scale(as.matrix(stackloss[, -4]))/sqrt(n-1)
t(scaled.X) %*% scaled.X
```

```
##             Air.Flow Water.Temp Acid.Conc.
## Air.Flow   1.0000000  0.7818523  0.5001429
## Water.Temp 0.7818523  1.0000000  0.3909395
## Acid.Conc. 0.5001429  0.3909395  1.0000000
```

```r
data = list(Y = stackloss$stack.loss, X=scaled.X, p=ncol(scaled.X))
data$n = n   #check

data$scales = attr(scaled.X, "scaled:scale")*sqrt(n-1) # fix scale
data$Xbar = attr(scaled.X, "scaled:center")

# define a function that returns the Model
  rr.model = function() {
    a  <- 9
    shape <- a/2
    delta <- 6
    delta.shape <- delta / 2

    for (i in 1:n) {
      mu[i] <- alpha0 + inprod(X[i,], alpha)
      lambda[i] ~ dgamma(shape, shape)
      prec[i] <- phi*lambda[i]
      Y[i] ~ dnorm(mu[i], prec[i])
    }
    phi ~ dgamma(1.0E-6, 1.0E-6)
    alpha0 ~ dnorm(0, 1.0E-6)

    for (j in 1:p) {
      prec.beta[j] <- lambda.beta[j]*phi
      alpha[j] ~ dnorm(0, prec.beta[j])
      beta[j] <- alpha[j]/scales[j]
      lambda.beta[j] ~ dgamma(delta.shape, delta.shape)
    }

    beta0 <- alpha0 - inprod(beta[1:p], Xbar)
```

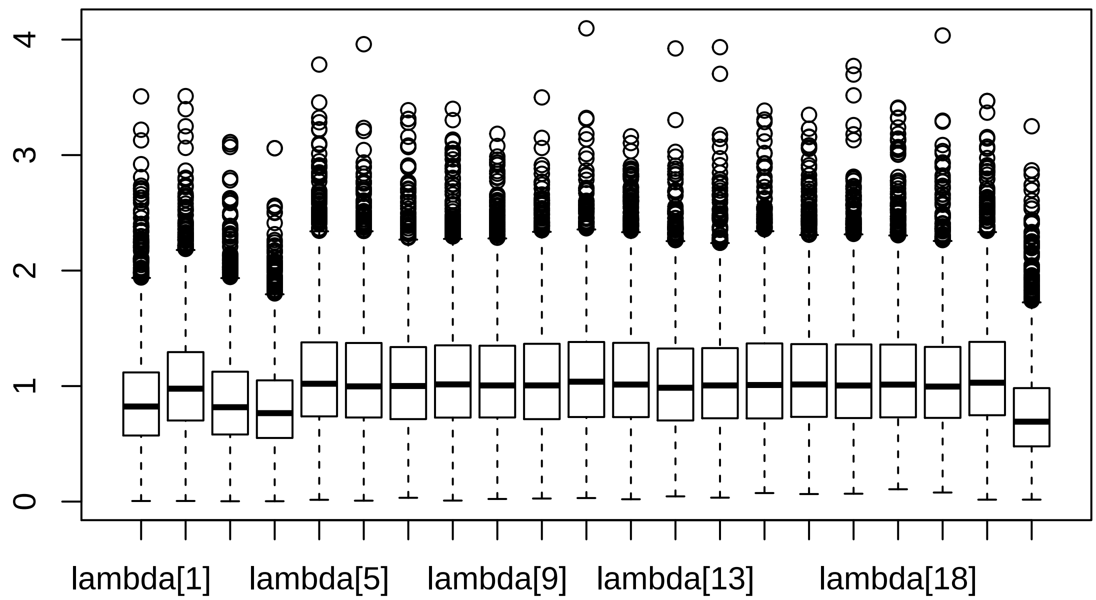
```r
    sigma <- pow(phi, -.5)
  }


  parameters = c("beta0", "beta", "sigma","lambda.beta", "lambda")


  bf.sim = jags(data, inits=NULL, par=parameters, model=rr.model,  n.iter=10000)
```

```
## module glm loaded

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 21
##    Unobserved stochastic nodes: 29
##    Total graph size: 242
##
## Initializing model
```

```r
bf.bugs = as.mcmc(bf.sim$BUGSoutput$sims.matrix)  # create an MCMC object

apply(bf.bugs,2,quantile,c(.025,.975))
```

```
##         beta[1]     beta[2]    beta[3]     beta0 deviance lambda[1]
## 2.5%  0.181348 -0.01383762 -0.3009032 -63.86155 103.1161 0.2648201
## 97.5% 1.073836  2.03347962  0.3112438 -14.55279 132.2432 1.8876991
##       lambda[2] lambda[3] lambda[4] lambda[5] lambda[6] lambda[7]
## 2.5%  0.3247252 0.2748706 0.2489955 0.3651464 0.3368616 0.3645663
## 97.5% 2.1785612 1.8933829 1.7574218 2.2369426 2.2267400 2.1835063
##       lambda[8] lambda[9] lambda[10] lambda[11] lambda[12] lambda[13]
## 2.5%  0.3504435 0.3619184    0.34479  0.3390623  0.3360589  0.3189446
## 97.5% 2.2407718 2.2418485    2.19484  2.2368247  2.2710731  2.1462323
##       lambda[14] lambda[15] lambda[16] lambda[17] lambda[18] lambda[19]
## 2.5%   0.3505399  0.3492846  0.3392411  0.3437709  0.3500109  0.3627427
## 97.5%  2.1804368  2.1976709  2.2922977  2.2565627  2.2661423  2.1941398
##       lambda[20] lambda[21] lambda.beta[1] lambda.beta[2] lambda.beta[3]
## 2.5%   0.3709288  0.2201934     0.01843818     0.07725768      0.2596854
## 97.5%  2.2329293  1.8124164     1.04135472     1.83354099      2.5234883
##          sigma
## 2.5%  2.734173
## 97.5% 6.808552
```

```r
boxplot(as.matrix(bf.bugs[,c(6:26)]))
```



The values that corresponded with outliers from our previous analysis are generally lower. However, they aren't necessarily as low as I was anticipating.

```
a.vec = c(5,10,15,20)
delta.vec = c(2,7,12,17)
nreps = 3000
mcmc.array = array(0, c(4,nreps,30))

for(i in 1:length(a.vec)){
# Create a data list with inputs for JAGS
a = a.vec[i]
delta = delta.vec[i]
n = nrow(stackloss)
## scale X such that X^TX has ones on the diagonal;
## scale divides by the standard deviation so we need
## to divide by the sqrt(n-1)
scaled.X = scale(as.matrix(stackloss[, -4]))/sqrt(n-1)
t(scaled.X) %*% scaled.X
data = list(Y = stackloss$stack.loss, X=scaled.X, p=ncol(scaled.X))
data$n = n   #check
data$a = a
data$delta = delta

data$scales = attr(scaled.X, "scaled:scale")*sqrt(n-1) # fix scale
data$Xbar = attr(scaled.X, "scaled:center")

# define a function that returns the Model
```

```r
  rr.model = function() {
    shape <- a/2
    delta.shape <- delta / 2

    for (i in 1:n) {
      mu[i] <- alpha0 + inprod(X[i,], alpha)
      lambda[i] ~ dgamma(shape, shape)
      prec[i] <- phi*lambda[i]
      Y[i] ~ dnorm(mu[i], prec[i])
    }
    phi ~ dgamma(1.0E-6, 1.0E-6)
    alpha0 ~ dnorm(0, 1.0E-6)

    for (j in 1:p) {
      prec.beta[j] <- lambda.beta[j]*phi
      alpha[j] ~ dnorm(0, prec.beta[j])
      beta[j] <- alpha[j]/scales[j]
      lambda.beta[j] ~ dgamma(delta.shape, delta.shape)
    }

    beta0 <- alpha0 - inprod(beta[1:p], Xbar)
    sigma <- pow(phi, -.5)
  }


  parameters = c("beta0", "beta", "sigma","lambda.beta", "lambda")


  bf.sim = jags(data, inits=NULL, par=parameters, model=rr.model,  n.iter=10000)


  bf.bugs = as.mcmc(bf.sim$BUGSoutput$sims.matrix)  # create an MCMC object

  mcmc.array[i,,] = bf.bugs
}
```

```
## module glm loaded

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 21
##    Unobserved stochastic nodes: 29
##    Total graph size: 237
##
## Initializing model
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 21
##    Unobserved stochastic nodes: 29
```

```
##    Total graph size: 237
##
## Initializing model
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 21
##    Unobserved stochastic nodes: 29
##    Total graph size: 237
##
## Initializing model
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 21
##    Unobserved stochastic nodes: 29
##    Total graph size: 237
##
## Initializing model
```

```r
first = mcmc.array[1,,]
second = mcmc.array[2,,]
third = mcmc.array[3,,]
fourth = mcmc.array[4,,]
apply(first,2,quantile,c(.025,.975))
```

```
##            [,1]        [,2]       [,3]      [,4]      [,5]      [,6]
## 2.5%  0.4474152 -0.07995619 -0.2711342 -60.71702  94.62811 0.1465559
## 97.5% 1.1464569  1.63745360  0.1640278 -23.47889 120.51504 2.3007144
##            [,7]      [,8]       [,9]     [,10]    [,11]     [,12]
## 2.5%  0.2178725 0.1180064 0.08856112 0.2444056 0.206274 0.2117897
## 97.5% 2.6099020 2.0605983 1.57786198 2.7055657 2.638736 2.7052428
##           [,13]    [,14]     [,15]     [,16]    [,17]     [,18]    [,19]
## 2.5%  0.2514063 0.210805 0.2277814 0.2278584 0.227773 0.1934357 0.211998
## 97.5% 2.9731424 2.727078 2.7688753 2.8160230 2.629464 2.3510452 2.714619
##           [,20]     [,21]     [,22]     [,23]     [,24]     [,25]
## 2.5%  0.2197606 0.2480727 0.2459721 0.2308512 0.2211305 0.2321218
## 97.5% 2.6992215 2.8895144 2.7429954 2.8526767 2.6846879 2.6325990
##            [,26]       [,27]      [,28]      [,29]    [,30]
## 2.5%  0.05634865 0.001192157 0.01197879 0.06659488 1.884774
## 97.5% 1.25276837 0.150394784 2.51865787 3.76747599 4.711938
```

```r
apply(second,2,quantile,c(.025,.975))
```

```
##            [,1]       [,2]       [,3]      [,4]     [,5]      [,6]
## 2.5%  0.1642209 0.01724267 -0.3015438 -65.45926 104.7391 0.2907439
## 97.5% 1.0422161 2.14008291  0.3398584 -13.41540 133.4926 1.8399362
##            [,7]      [,8]     [,9]     [,10]     [,11]     [,12]    [,13]
## 2.5%  0.3546388 0.2928907 0.282716 0.3617638 0.3495033 0.3719705 0.362615
## 97.5% 2.0839929 1.7928596 1.796316 2.1823122 2.0940096 2.1204549 2.120535
##           [,14]     [,15]     [,16]     [,17]     [,18]     [,19]
```

```
## 2.5%  0.3755576 0.3624019 0.3632469 0.3733862 0.3904789 0.3627028
## 97.5% 2.1678893 2.0805231 2.1232457 2.1910691 2.1276077 2.1327031
##           [,20]     [,21]     [,22]     [,23]     [,24]     [,25]    [,26]
## 2.5%  0.3859523 0.3614737 0.3755514 0.3617568 0.3693363 0.3523097 0.255307
## 97.5% 2.1650482 2.1949358 2.1425765 2.1338302 2.1156969 2.1839233 1.812269
##            [,27]      [,28]     [,29]    [,30]
## 2.5%  0.02829604 0.09673233 0.2720399 2.886535
## 97.5% 1.08218778 1.74464346 2.4036614 7.270100
```

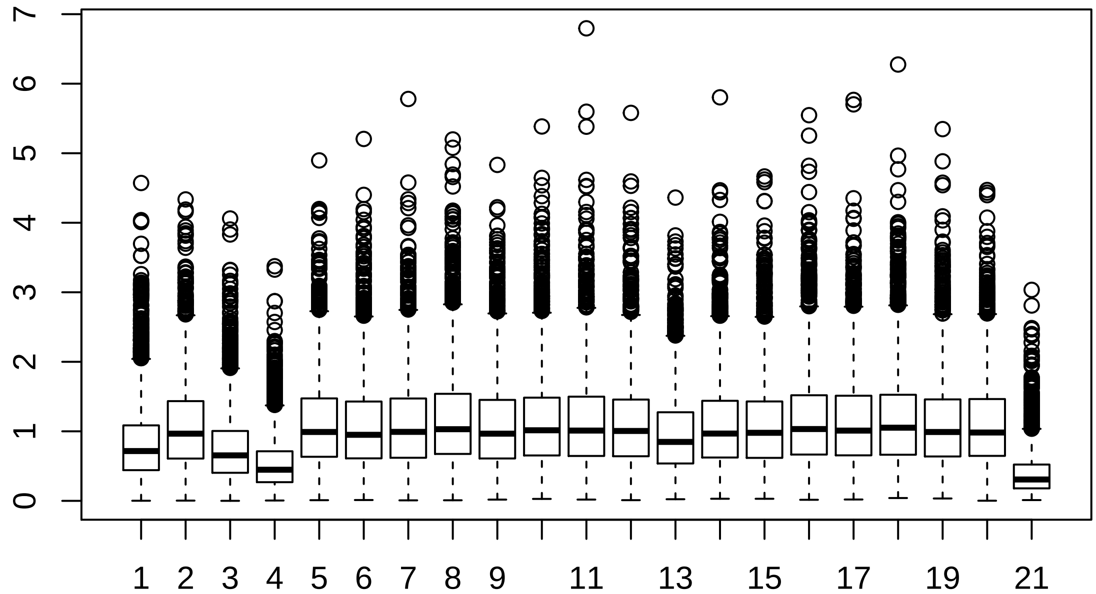
```
apply(third,2,quantile,c(.025,.975))
```

```
##            [,1]      [,2]       [,3]       [,4]     [,5]      [,6]
## 2.5%  0.1224153 0.1417754 -0.2828927 -68.627235 111.1768 0.3790558
## 97.5% 0.8960135 2.0178586  0.4191964  -7.060357 138.6068 1.6391419
##            [,7]      [,8]     [,9]    [,10]     [,11]     [,12]     [,13]
## 2.5%  0.4295628 0.3887783 0.408609 0.434376 0.4648468 0.4602179 0.4487115
## 97.5% 1.8122669 1.6692538 1.752913 1.896194 1.9006816 1.8796755 1.8655665
##           [,14]    [,15]     [,16]     [,17]     [,18]    [,19]     [,20]
## 2.5%  0.4456763 0.442547 0.4480747 0.4382497 0.4630359 0.451270 0.4320611
## 97.5% 1.8497883 1.937476 1.9640736 1.8740695 1.9133283 1.891655 1.9088033
##           [,21]     [,22]     [,23]     [,24]     [,25]     [,26]
## 2.5%  0.4484011 0.4477792 0.4470546 0.4495215 0.4557973 0.4069173
## 97.5% 1.8986917 1.9190865 1.8632476 1.8697184 1.8788665 1.7511668
##           [,27]     [,28]     [,29]    [,30]
## 2.5%  0.1200113 0.2040908 0.3899878 3.804464
## 97.5% 1.4054388 1.6159028 1.9831746 8.242834
```

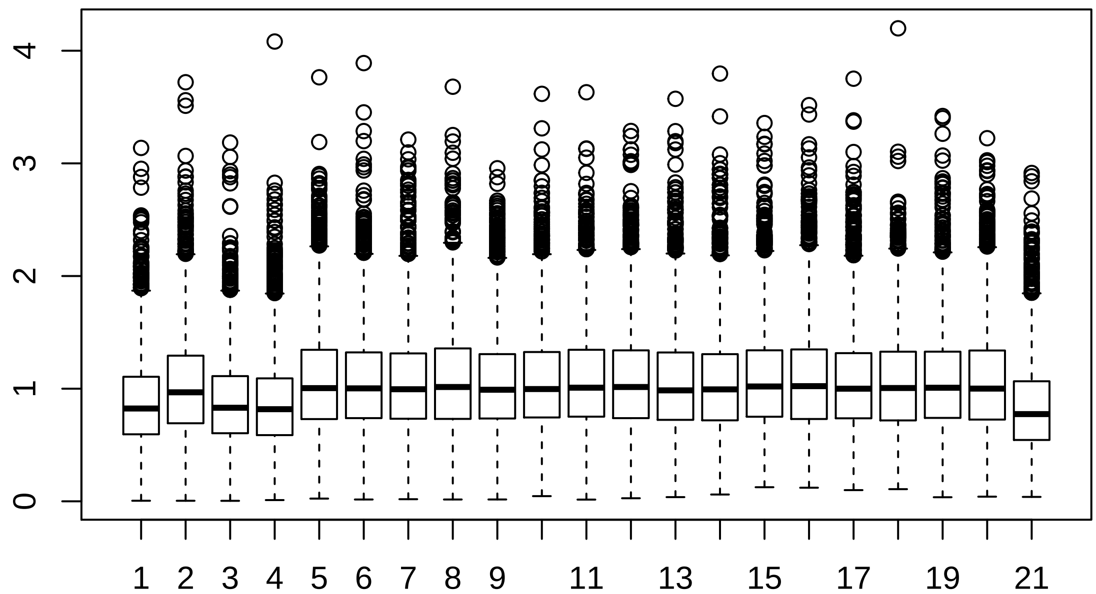
```
apply(fourth,2,quantile,c(.025,.975))
```

```
##            [,1]      [,2]       [,3]       [,4]     [,5]      [,6]
## 2.5%  0.1100597 0.1763736 -0.2919183 -69.023653 114.3520 0.4411753
## 97.5% 0.8018687 1.9729589  0.4570459  -4.362205 139.9185 1.5827504
##           [,7]      [,8]      [,9]     [,10]     [,11]     [,12]     [,13]
## 2.5%  0.507975 0.4337737 0.4801587 0.5119817 0.5029083 0.5066742 0.5143207
## 97.5% 1.711326 1.6053888 1.6438983 1.7956507 1.7448596 1.7287675 1.7522078
##           [,14]     [,15]     [,16]     [,17]     [,18]    [,19]     [,20]
## 2.5%  0.4964925 0.5046954 0.5029934 0.5139379 0.5080781 0.513897 0.5003512
## 97.5% 1.7113436 1.7464695 1.8115251 1.7736927 1.7350794 1.769228 1.7446216
##           [,21]     [,22]     [,23]     [,24]    [,25]    [,26]    [,27]
## 2.5%  0.4958172 0.5116327 0.4951016 0.5104638 0.502379 0.473089 0.221777
## 97.5% 1.7302052 1.7569632 1.7645631 1.7657160 1.735113 1.680046 1.423883
##          [,28]     [,29]    [,30]
## 2.5%  0.302976 0.4727828 4.190547
## 97.5% 1.515969 1.7729833 8.624839
```

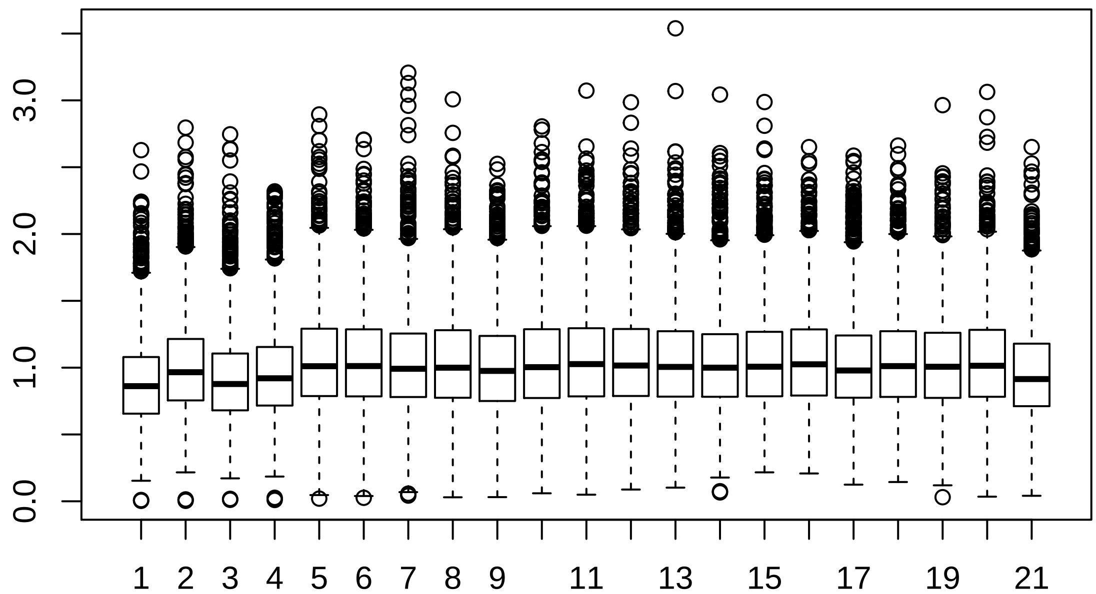
```
boxplot(as.matrix(first[,c(6:26)]))
```



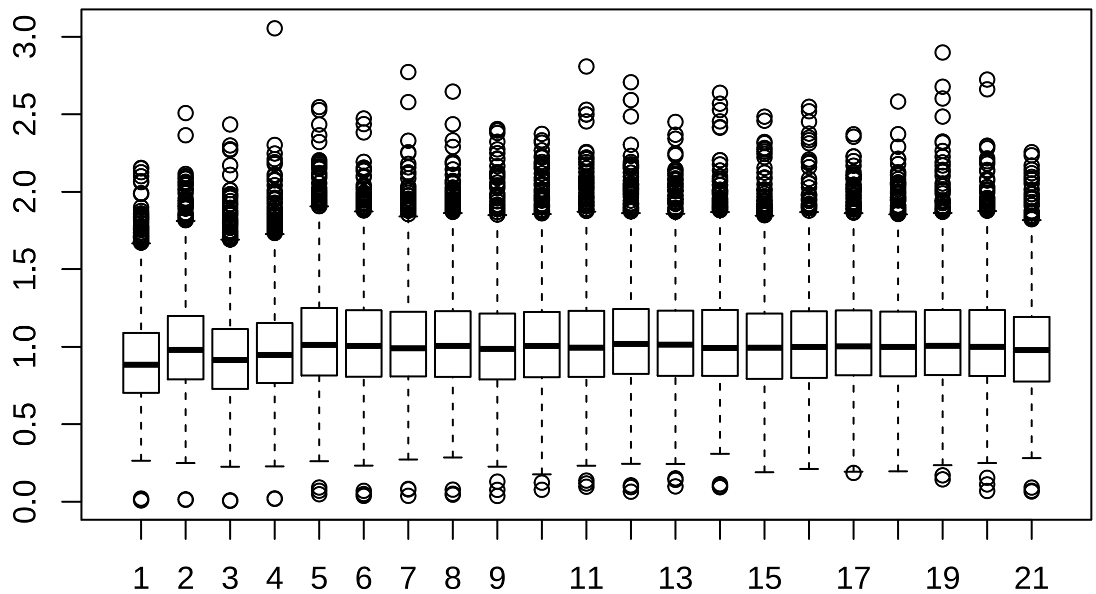

```
boxplot(as.matrix(second[,c(6:26)]))
```




```r
boxplot(as.matrix(third[,c(6:26)]))
```




```r
boxplot(as.matrix(fourth[,c(6:26)]))
```



It seems clear that the value of λ is related to whether or not something is an outlier. The lower values of λ generally are indicative of the presence of outlieras. However, it is important to note that this also changes with our values of a and δ. When these are larger, then the values of λ generally increase. So we could hypothetically tune these paramters to adjust for how probable we we think there are outliers in the data and how much they are affecting our β values. If we are sure that there are outliers, then we can se a and δ lower because that makes it more likely that there are outliers. Now that since the values of λ change based on a and δ, it is hard to know what are true outliers.

I now compare this methodology with MC3.REG and BAS. We did this in the last assignment.

```
attach(stackloss)
```

```
## The following object is masked _by_ .GlobalEnv:
##
##     stack.loss
```

```
## The following object is masked from package:datasets:
##
##     stack.loss
```

```
stack.MC3= MC3.REG(stack.loss, stackloss[,-4]
  ,num.its=10000, outliers=TRUE, M0.out=rep(FALSE, 21), outs.list=1:21, M0.var=rep(TRUE, 3))

summary(stack.MC3)
```

```
##
## Call:
## MC3.REG(all.y = stack.loss, all.x = stackloss[, -4], num.its = 10000,     M0.var = rep(TRUE, 3), M0.o
##
```

```
## Model parameters: PI = 0.1 K = 7 nu = 0.2 lambda = 0.1684 phi = 9.2
##
##   2064  models were selected
##  Best  5  models (cumulative posterior probability =  0.4466 ):
##
##                 prob     model 1  model 2  model 3  model 4  model 5
## variables
##   Air.Flow    0.99999   x        x        x        x        x
##   Water.Temp  0.61221   x        .        x        x        x
##   Acid.Conc.  0.05074   .        .        .        .        .
## outliers
##   1           0.49451   x        .        .        x        .
##   2           0.06082   .        .        .        .        .
##   3           0.51612   x        .        .        x        .
##   4           0.91001   x        x        x        x        .
##   5           0.01859   .        .        .        .        .
##   6           0.02523   .        .        .        .        .
##   7           0.01831   .        .        .        .        .
##   8           0.01494   .        .        .        .        .
##   9           0.02156   .        .        .        .        .
##   10          0.01632   .        .        .        .        .
##   11          0.01519   .        .        .        .        .
##   12          0.02032   .        .        .        .        .
##   13          0.14548   .        .        .        x        .
##   14          0.06029   .        .        .        .        .
##   15          0.02040   .        .        .        .        .
##   16          0.01471   .        .        .        .        .
##   17          0.01666   .        .        .        .        .
##   18          0.01660   .        .        .        .        .
##   19          0.02498   .        .        .        .        .
##   20          0.04795   .        .        .        .        .
##   21          0.98557   x        x        x        x        x
##
## post prob             0.18451  0.13617  0.06913  0.03088  0.02587
```

```r
detach(stackloss)
n = nrow(stackloss)
stack.out = cbind(stackloss, diag(n))  #add indicators
BAS.stack.pois = bas.lm(stack.loss ~ .,
                   data=stack.out,
                   prior="hyper-g-n", a=3,
                   modelprior=tr.poisson(4, 15),
                   method="MCMC",
                   MCMC.iterations =50000)

BAS.stack.pois
```

```
##
## Call:
## bas.lm(formula = stack.loss ~ ., data = stack.out, prior = "hyper-g-n",     alpha = 3, modelprior = t
##
##
##  Marginal Posterior Inclusion Probabilities:
##  Intercept     Air.Flow  Water.Temp  Acid.Conc.         `1`         `2`
##    1.00000      1.00000     0.48765     0.06599     0.65254     0.14612
```

```
##        `3`          `4`          `5`          `6`          `7`          `8`
##    0.67775      0.98222      0.04373      0.05274      0.04140      0.04570
##        `9`         `10`         `11`         `12`         `13`         `14`
##    0.04387      0.04392      0.04079      0.05983      0.34026      0.16951
##       `15`         `16`         `17`         `18`         `19`         `20`
##    0.04713      0.03548      0.04271      0.04117      0.06611      0.11543
##       `21`
##    0.99740
```

```
t(summary(BAS.stack.pois))
```

```
##                   [,1]       [,2]       [,3]       [,4]       [,5]
## Intercept   1.00000  1.0000000  1.0000000  1.0000000  1.0000000
## Air.Flow    1.00000  1.0000000  1.0000000  1.0000000  1.0000000
## Water.Temp  1.00000  0.0000000  1.0000000  1.0000000  0.0000000
## Acid.Conc.  0.00000  0.0000000  0.0000000  0.0000000  0.0000000
## `1`         1.00000  1.0000000  1.0000000  1.0000000  1.0000000
## `2`         0.00000  0.0000000  0.0000000  1.0000000  0.0000000
## `3`         1.00000  1.0000000  1.0000000  1.0000000  1.0000000
## `4`         1.00000  1.0000000  1.0000000  1.0000000  1.0000000
## `5`         0.00000  0.0000000  0.0000000  0.0000000  0.0000000
## `6`         0.00000  0.0000000  0.0000000  0.0000000  0.0000000
## `7`         0.00000  0.0000000  0.0000000  0.0000000  0.0000000
## `8`         0.00000  0.0000000  0.0000000  0.0000000  0.0000000
## `9`         0.00000  0.0000000  0.0000000  0.0000000  0.0000000
## `10`        0.00000  0.0000000  0.0000000  0.0000000  0.0000000
## `11`        0.00000  0.0000000  0.0000000  0.0000000  0.0000000
## `12`        0.00000  0.0000000  0.0000000  0.0000000  0.0000000
## `13`        0.00000  1.0000000  1.0000000  0.0000000  0.0000000
## `14`        0.00000  0.0000000  0.0000000  0.0000000  0.0000000
## `15`        0.00000  0.0000000  0.0000000  0.0000000  0.0000000
## `16`        0.00000  0.0000000  0.0000000  0.0000000  0.0000000
## `17`        0.00000  0.0000000  0.0000000  0.0000000  0.0000000
## `18`        0.00000  0.0000000  0.0000000  0.0000000  0.0000000
## `19`        0.00000  0.0000000  0.0000000  0.0000000  0.0000000
## `20`        0.00000  0.0000000  0.0000000  0.0000000  0.0000000
## `21`        1.00000  1.0000000  1.0000000  1.0000000  1.0000000
## BF          1.00000  0.3337882  0.5684913  0.5460933  0.2002082
## PostProbs   0.08420  0.0281000  0.0274000  0.0263000  0.0253000
## R2          0.98920  0.9873000  0.9923000  0.9922000  0.9803000
## dim         7.00000  7.0000000  8.0000000  8.0000000  6.0000000
## logmarg    24.17879 23.0815428 23.6140221 23.5738258 22.5703939
```

The results seem comparable. However, it does seem more difficult to detect outliers with higher levels of a and δ because the values generally get bigger also. This seems like it could present a challenge. However, there does seem to be consensus that 1, 3, 4, and 21 are outliers

Exercise 4

```
# Create a data list with inputs for JAGS

n = nrow(stackloss)
scaled.X = scale(as.matrix(stackloss[, -4]))
```

```r
data = list(Y = stackloss$stack.loss, X=scaled.X, p=ncol(scaled.X))
data$n = n   #check

data$scales = attr(scaled.X, "scaled:scale")
data$Xbar = attr(scaled.X, "scaled:center")

# define a function that returns the Model
horseshoe.model = function() {
  for (i in 1:n) {
    mu[i] <- alpha0 + inprod(X[i,], alpha)
    Y[i] ~ dnorm(mu[i], phi)
  }
  phi ~ dgamma(1.0E-6, 1.0E-6)
  alpha0 ~ dnorm(0, 1.0E-6)

  for (j in 1:p) {
    prec.beta[j] <- phi/pow(tau[j],2)
    tau[j] ~ dt(0,1/lambda^2,1)%_%T(0,)

    alpha[j] ~ dnorm(0, prec.beta[j])
    beta[j] <- alpha[j]/scales[j]
  }
  lambda ~ dt(0,1,1)%_%T(0,)
  beta0 <- alpha0 - inprod(beta[1:p], Xbar)
  sigma <- pow(phi, -.5)
}


parameters = c("beta0", "beta", "sigma","tau")


horse.sim = jags(data, inits=NULL, par=parameters, model=horseshoe.model,  n.iter=10000)
```

```
## module glm loaded
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 21
##    Unobserved stochastic nodes: 9
##    Total graph size: 220
##
## Initializing model
```

```r
horse.bugs = as.mcmc(horse.sim$BUGSoutput$sims.matrix)  # create an MCMC object

apply(horse.bugs,2,quantile,c(.025,.075))
```

```
##        beta[1]   beta[2]    beta[3]     beta0 deviance    sigma    tau[1]
## 2.5% 0.4210710 0.3733296 -0.3763985 -62.71325 105.6788 2.434942 0.6799475
## 7.5% 0.4938663 0.6402042 -0.2950709 -57.31224 106.2452 2.607438 0.8890742
##         tau[2]     tau[3]
## 2.5% 0.3509496 0.01606183
## 7.5% 0.5352300 0.04817709
```

Exercise 5

All of these different analyses use different techniques for the same purpose of handling outliers. So, obviously some will have advantages over others. For example, we don't are not dealing with any type of outlier detection in the horseshoe. We could hypothetically change this by adding indicator functions to our design matrix, but currently I haven't done this.

However, the previous methods do deal with variable selection. Something that I note though is that under the the bounded influence, the interpretation can be a little bit more difficult. It does seem like a very plausible way to do it. The MC3reg was specifically designed for variable selection and outlier detection. Whereas we are coercing BAS into doing this. The frequentist methods that we've gone over, they can perform variable selection and outlier detection, but they are distinct operators. From the outlier detection, we can't get meaningful β values. Whereas under these approaches, we can get meaningful β values while doing outlier detection.