STA 602 HW 12

William Tirone

9.1

```
# load data
swim = as.matrix(read.table('http://www2.stat.duke.edu/~pdh10/FCBS/Exercises/swim.dat'))
```

a)

Referencing Hoff p. 154/155:

```
# data
# response is weeks
X = cbind(rep(1, 6), seq(1, 11, 2))
n = dim(X)[1]

# priors
B0 = c(23,0)
Sig0 = matrix(c(0.25,0,0,0.1),2,2,byrow = TRUE)
nu0 = 1
sig2.0 = 0.25

# iterations
S = 5000

# initial values
B = B0
sig2 = sig2.0

#output
beta_out = matrix(NA,nrow = S,ncol = 2)
```

```r
sig2_out = c()

for (j in 1:4){

  # data for jth swimmer
  y = swim[j,]

  # gibb's sample linear model
  for (i in 1:S) {

    B.curr = B
    sig2.curr = sig2

    # updating Beta
    V = solve(solve(Sig0)  +  (t(X) %*% X) / sig2.curr)
    m = V %*% (solve(Sig0) %*% B0 + (t(X) %*% y)/sig2.curr )
    B = mvrnorm(1,m,V)

    # updating sigma2
    SSR = t(y - X %*% B) %*% (y - X %*% B)
    sig2 = 1 / rgamma(1, (nu0 + n)/2, (nu0 * sig2.0 + SSR)/2)

    # store values
    beta_out[i,] = B
    sig2_out = append(sig2_out, sig2)

  }

  x_predict = c(1,13)
  y_predict = rnorm(S, beta_out %*% x_predict, sqrt(sig2_out))

  print(mean(y_predict))
}
```

```
[1] 22.62962
[1] 23.58634
[1] 22.87546
[1] 23.45656
```

b)

Since we want the fastest swimmer, using the predictive distribution (this is just a normal model since we have a MVN sampling and inverse gamma prior) we see that the first swimmer has the fastest mean time at 22.63978. So we will choose swimmer 1 to race.

9.2

```
az = as.matrix(read.table('http://www2.stat.duke.edu/~pdh10/FCBS/Exercises/azdiabetes.dat'
```

a)

Hoff p. 157

We can sample from $p(\sigma^2|y, X)$ and $p(\beta|y, X, \sigma^2)$ directly and can use vanilla MC to do so.

```
# make subset numeric
dat = apply(az[,-8], 2, as.numeric)

# data
X = dat[,-2]
y = dat[,2] # response, glucose
n = dim(X)[1]
p = dim(X)[2]
I = diag(1,n,n)

# priors
g = n
nu0 = 2
sig2.0 = 1

# samples
S = 5000

# sample sigma^2
SSRg = t(y) %*% (I - (g/(g+1)) * X %*% solve(t(X) %*% X) %*% t(X)) %*% y
sig2 = 1 / rgamma(S, (nu0 + n)/2, (nu0 * sig2.0 + SSRg)/2)

# sample Beta
Vb = (g/(g+1)) * solve(t(X) %*% X)
```

```r
Eb = Vb %*% t(X) %*% y
E = matrix(rnorm(S*p,0,sqrt(sig2)),S,p)
beta = t(t(E %*% chol(Vb)) + c(Eb))
```

Confidence Regions for the variables are below:

```r
cat("Sigma^2 95% Confidence Region: ",quantile(sig2,c(0.025,0.975)))
```

```
Sigma^2 95% Confidence Region:  796.5372 1011.236
```

```r
custom_quant = function(X) {
  quantile(X, c(0.025,0.975), na.rm = T)
}

apply(beta,2,custom_quant)
```

```
            npreg        bp       skin       bmi       ped       age
2.5%  -1.96011964 0.4106724 -0.2305393 0.7905132  5.939806 0.6364316
97.5%  0.02561324 0.7863974  0.4110757 1.7035482 20.678724 1.2756200
```

Comparing the MC results to a standard `lm()` fit out of curiosity. The intervals capture most of the betas below fairly well, though some do not if they have large standard errors.

```r
test_linear = lm(glu ~ ., data.frame(dat))
summary(test_linear)
```

```
Call:
lm(formula = glu ~ ., data = data.frame(dat))

Residuals:
    Min      1Q  Median      3Q     Max 
-71.556 -20.260  -2.648  18.396  86.008 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  52.3052     8.6023   6.080 2.31e-09 ***
npreg        -0.6571     0.4910  -1.338  0.18138    
bp            0.2053     0.1134   1.811  0.07077 .  
```

```
skin          0.1926     0.1571   1.226  0.22084
bmi           0.6444     0.2469   2.610  0.00931 **
ped          10.5484     3.6752   2.870  0.00427 **
age           0.7667     0.1587   4.831 1.79e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 28.69 on 525 degrees of freedom
Multiple R-squared:  0.1529,    Adjusted R-squared:  0.1432
F-statistic: 15.79 on 6 and 525 DF,  p-value: < 2.2e-16
```

b)

```r
# calculate log(p(y|X))
lpy.X <- function(y, X, g=length(y), nu0=1,
                  s20=try(summary(lm(y ~ 1+X))$sigma^2,
                          silent=TRUE))
  {
    n <- dim(X)[1];p <- dim(X)[2]
    if(p==0){Hg <- 0;s20 <- mean(y^2)}
      if(p>0){Hg <- (g/(g+1)) * X %*% solve(t(X) %*% X) %*% t(X)}
        SSRg <- t(y) %*% (diag(1,nrow=n)-Hg) %*% y
        -.5*(n*log(pi) + p*log(1 + g) + (nu0 + n)*log(nu0*s20 + SSRg)-
        nu0*log(nu0*s20)) +
        lgamma((nu0 + n)/2)-lgamma(nu0/2)
  }

# set up
z <- rep(1,dim(X)[2])
lpy.c <- lpy.X(y,X[,z==1,drop=FALSE])
S <- 10000
Z <- matrix(NA,S,dim(X)[2])

beta_output = data.frame()


# Gibb's sampler
for(s in 1:S)
{
  for(j in sample(1:dim(X)[2]))
    {
```

```r
      zp <- z;zp[j] <- 1-zp[j]
      lpy.p <- lpy.X(y,X[,zp==1,drop=FALSE])
      r <- (lpy.p-lpy.c)*(-1)^(zp[j]==0)
      z[j] <- rbinom(1,1,1/(1 + exp(-r)))
      if(z[j]==zp[j]){lpy.c <- lpy.p}
    }

  # create an X_z for Z = 1
  X_z = X[,c(which(z==1))]

  # sample sigma^2
  SSRg_z = t(y) %*% (I - (g/(g+1)) *
                       X_z %*% solve(t(X_z) %*% X_z) %*% t(X_z)) %*% y
  sig2 = 1 / rgamma(1, (nu0 + n)/2, (nu0 * sig2.0 + SSRg_z)/2)

  # sample Beta
  Vb = g * sig2 * solve(t(X_z) %*% X_z)
  m = matrix(0,dim(Vb)[1],1)
  beta = mvrnorm(1,m,Vb)

  # save results
  Z[s,] <- z

  beta_output = dplyr::bind_rows(beta_output,beta)
}
```

Below, finding $p(\beta_j \neq 0|y)$ and confidence intervals

I'm not entirely sure I did this correctly but my reasoning is that if the model selected NA for that particular beta, then the posterior probability would just be calculated in the usual Monte Carlo way of calculating the average number of values that are not zero. There are no NAs for the "bp" column, for example, so the probability that it is not zero is equal to 1, so we should always use that column.

```r
# number of NA values
1 - (apply(apply(beta_output, 2, is.na),2,sum) / 1000)
```

```
   bp    bmi    ped    age   skin  npreg
1.000  1.000  0.569  1.000 -8.555 -6.898
```

Confidence Intervals: These look quite different compared to part a. The variables that look the most different have a very low probability of being selected (computed above); for example, skin has a huge interval but a very small one in part a, the same with npreg. I think this expresses the model averaging suggesting these shouldn't be used.

```
apply(beta_output,2,custom_quant)
```

```
             bp       bmi       ped       age      skin     npreg
2.5%  -4.373122 -8.239379 -173.6413 -6.199707 -6.503620 -21.77233
97.5%  4.312382  8.371234  171.4095  6.190989  6.329778  23.24580
```

9.3

a)

Adapting code from 9.2 a), we can use vanilla MC.

It looks like M, Ed, U2, Ineq, and Prob (we can view the definitions of these with ?UScrime if desired) reach statistical significance with a linear regression with lm. The relationships for all of these are positive, except for Prob which has a negative relationship.

From a quick visual inspection, the marginal posterior means are very close to the OLS estimates. The posterior confidence intervals do not have an equivalent frequentist interpretation, but allow us to make probabilistic statements about the coefficients.

```
# make subset numeric
dat = apply(crime, 2, as.numeric)

# data
X = dat[,-1]
y = dat[,1] # response
n = dim(X)[1]
p = dim(X)[2]
I = diag(1,n,n)

# priors
g = n
nu0 = 2
sig2.0 = 1

# samples
S = 5000
```

```r
# sample sigma^2
SSRg = t(y) %*% (I - (g/(g+1)) * X %*% solve(t(X) %*% X) %*% t(X)) %*% y
sig2 = 1 / rgamma(S, (nu0 + n)/2, (nu0 * sig2.0 + SSRg)/2)

# sample Beta
Vb = (g/(g+1)) * solve(t(X) %*% X)
Eb = Vb %*% t(X) %*% y
E = matrix(rnorm(S*p,0,sqrt(sig2)),S,p)
beta = t(t(E %*% chol(Vb)) + c(Eb))
```

Obtaining marginal posterior means and 95% CI:

```r
print("Posterior Beta Means")
```

```
[1] "Posterior Beta Means"
```

```r
as.matrix(apply(beta,2,mean))
```

```
              [,1]
M     0.2821819335
So    0.0002243641
Ed    0.5325152615
Po1   1.4378690303
Po2  -0.7635463390
LF   -0.0660507181
M.F   0.1289176846
Pop  -0.0712688689
NW    0.1075115475
U1   -0.2620112138
U2    0.3582710907
GDP   0.2378577192
Ineq  0.7085405408
Prob -0.2787253183
Time -0.0572680545
```

```r
print("95% CI")
```

```
[1] "95% CI"
```

```r
apply(beta,2,custom_quant)
```

```
                M          So        Ed          Po1         Po2          LF
2.5%  0.04007692 -0.3255593 0.220881 -0.01538701 -2.2882085 -0.3409656
97.5% 0.52656390  0.3379026 0.866339  2.89263278  0.7564877  0.2069706
              M.F         Pop         NW          U1          U2         GDP
2.5%  -0.1468858 -0.3009984 -0.1966579 -0.61252541 0.03412174 -0.2231677
97.5%  0.4011395  0.1581005  0.4180287  0.08617739 0.68785556  0.7110082
            Ineq         Prob        Time
2.5%  0.2985394 -0.51948805 -0.2858403
97.5% 1.1250785 -0.04325157  0.1710093
```

Now fitting OLS:

```r
OLS_fit = lm(y ~ ., data.frame(crime))
summary(OLS_fit)
```

```
Call:
lm(formula = y ~ ., data = data.frame(crime))

Residuals:
     Min       1Q   Median       3Q      Max
-1.02571 -0.26223 -0.01598  0.29013  1.33038

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.0004581  0.0789333  -0.006  0.99541
M            0.2865181  0.1357981   2.110  0.04304 *
So          -0.0001140  0.1840530  -0.001  0.99951
Ed           0.5445141  0.1796106   3.032  0.00488 **
Po1          1.4716211  0.8166435   1.802  0.08127 .
Po2         -0.7817801  0.8515935  -0.918  0.36570
LF          -0.0659646  0.1534476  -0.430  0.67025
M.F          0.1312980  0.1551792   0.846  0.40398
Pop         -0.0702919  0.1269186  -0.554  0.58367
NW           0.1090567  0.1719187   0.634  0.53051
U1          -0.2705364  0.1966309  -1.376  0.17872
U2           0.3687303  0.1798586   2.050  0.04889 *
GDP          0.2380595  0.2589534   0.919  0.36503
```

```
Ineq         0.7262920  0.2341502   3.102  0.00408 **
Prob        -0.2852264  0.1337912  -2.132  0.04105 *
Time        -0.0615769  0.1310241  -0.470  0.64167
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5411 on 31 degrees of freedom
Multiple R-squared:  0.8029,    Adjusted R-squared:  0.7075
F-statistic: 8.418 on 15 and 31 DF,  p-value: 3.59e-07
```

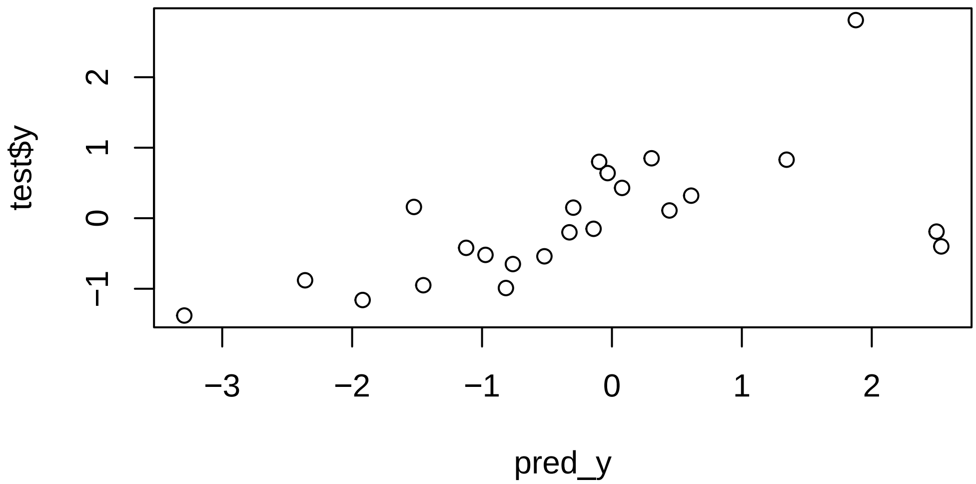
b)

Looking at the next several cells, the lm fit achieves an MSE of .61 compared to .47 for the Bayesian approach with a g prior. I would guess the Bayesian model does a better job because it more accurately captures whether or not a covariate should be used by using probability values rather than the frequentist approach.

i)

Predicted values are plotted compared to y_{te} below.

```r
# reload to be safe
crime = read.table('http://www2.stat.duke.edu/~pdh10/FCBS/Exercises/crime.dat',
             header = T)

y = crime[,'y']
X = crime[,-1] # not y

index = createDataPartition(y,1)


train = crime[index$Resample1,]
test = crime[-index$Resample1,]

train_fit = lm(y ~ ., data = train)
pred_y = predict(train_fit, newdata = test)

plot(pred_y, test$y)
```



Looking at coefficients for lm:

```
summary(train_fit)$coefficients
```

```
               Estimate Std. Error    t value   Pr(>|t|)
(Intercept) -0.09836741  0.2460777 -0.3997413 0.69980753
M            0.75624760  0.4041896  1.8710218 0.09825257
So           0.01341311  0.4785849  0.0280266 0.97832747
Ed           0.61260794  0.3814031  1.6061954 0.14689821
Po1         -0.95724052  2.1122111 -0.4531936 0.66244868
Po2          1.84718474  2.2778902  0.8109191 0.44086765
LF           0.49491850  0.4650844  1.0641477 0.31832406
M.F         -0.51662276  0.4697674 -1.0997417 0.30343377
Pop         -0.19915004  0.2698457 -0.7380146 0.48159083
NW           0.17824472  0.3894757  0.4576530 0.65937497
U1          -0.61420970  0.6522199 -0.9417218 0.37389857
U2           0.99557717  0.5193219  1.9170714 0.09152797
GDP          0.05664215  0.5525512  0.1025102 0.92087466
Ineq         0.43772968  0.5491789  0.7970621 0.44842239
Prob        -0.34570141  0.3045833 -1.1349979 0.28923565
Time        -0.17875287  0.2570136 -0.6954997 0.50643906
```

Now computing MSE:

```
mean((test$y - pred_y)^2)
```

```
[1] 1.275324
```

ii) Now with the g prior:

```
# make subset numeric
dat = apply(train, 2, as.numeric)

# data
X = dat[,-1]
y = dat[,1] # response
n = dim(X)[1]
p = dim(X)[2]
I = diag(1,n,n)

# priors
g = n
nu0 = 2
sig2.0 = 1

# samples
S = 5000

# sample sigma^2
SSRg = t(y) %*% (I - (g/(g+1)) * X %*% solve(t(X) %*% X) %*% t(X)) %*% y
sig2 = 1 / rgamma(S, (nu0 + n)/2, (nu0 * sig2.0 + SSRg)/2)

# sample Beta
Vb = (g/(g+1)) * solve(t(X) %*% X)
Eb = Vb %*% t(X) %*% y
E = matrix(rnorm(S*p,0,sqrt(sig2)),S,p)
beta = t(t(E %*% chol(Vb)) + c(Eb))

print("posterior means : ")
```

```
[1] "posterior means : "
```

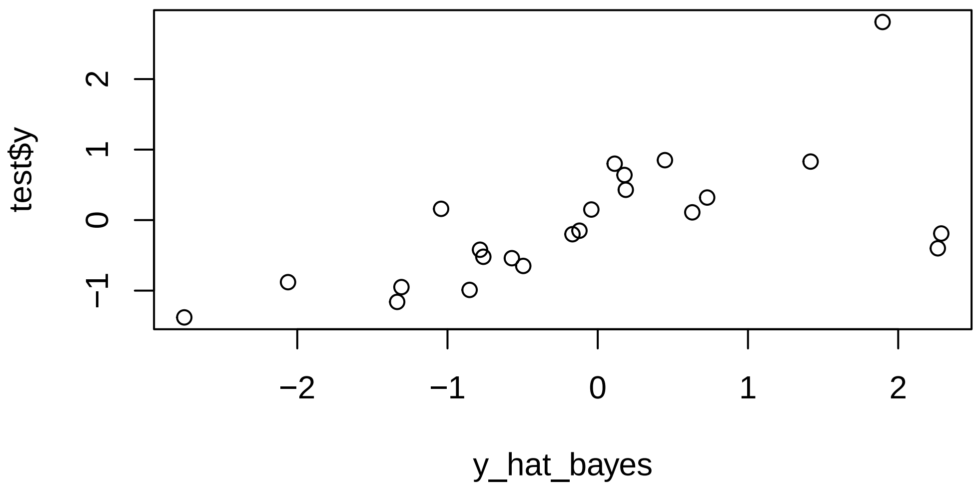
```
beta_hat_bayes = as.matrix(apply(beta,2,mean))
beta_hat_bayes
```

```
           [,1]
M     0.6740100
So    0.1029460
```

```
Ed     0.5577938
Po1   -0.6695758
Po2    1.5025529
LF     0.4572066
M.F   -0.4799568
Pop   -0.2053222
NW     0.1339545
U1    -0.4202117
U2     0.8455461
GDP    0.1121805
Ineq   0.4681924
Prob  -0.3837733
Time  -0.1608419
```

Now obtaining the predictions $\hat{y}$

```r
test_bayes = apply(test[,-1], 2, as.numeric)
y_hat_bayes = test_bayes %*% beta_hat_bayes # predictions for data

plot(y_hat_bayes, test$y)
```



Prediction (Mean Squared) error:

```r
mean((test$y - y_hat_bayes)^2)
```

```
[1] 0.9090856
```

c)

With many repetitions, the results look very similar and both methods produce extremely similar results and the fit between the two is almost linear.

```
# helper function for g prior
# should have done this earlier
bayes_regression = function(data){

  # make subset numeric
  dat = apply(data, 2, as.numeric)

  # data
  X = dat[,-1]
  y = dat[,1] # response
  n = dim(X)[1]
  p = dim(X)[2]
  I = diag(1,n,n)

  # priors
  g = n
  nu0 = 2
  sig2.0 = 1

  # samples
  S = 5000

  # sample sigma^2
  SSRg = t(y) %*% (I - (g/(g+1)) * X %*% solve(t(X) %*% X) %*% t(X)) %*% y
  sig2 = 1 / rgamma(S, (nu0 + n)/2, (nu0 * sig2.0 + SSRg)/2)

  # sample Beta
  Vb = (g/(g+1)) * solve(t(X) %*% X)
  Eb = Vb %*% t(X) %*% y
  E = matrix(rnorm(S*p,0,sqrt(sig2)),S,p)
  beta = t(t(E %*% chol(Vb)) + c(Eb))

  #compute means
  beta_m = as.matrix(apply(beta,2,mean))

  return(beta_m)
```

```
}

bayes_regression(train)
```

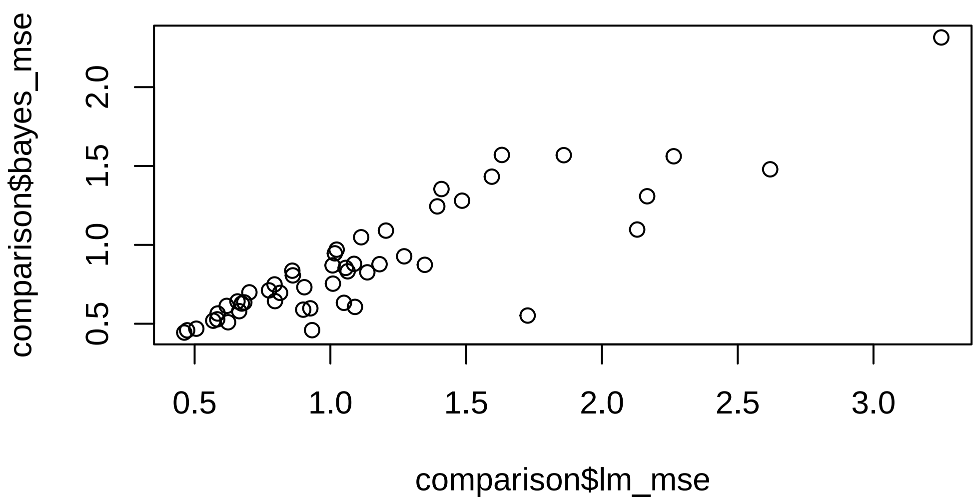
```
           [,1]
M     0.6819537
So    0.1006151
Ed    0.5616230
Po1  -0.6846249
Po2   1.5157421
LF    0.4632649
M.F  -0.4852544
Pop  -0.2026616
NW    0.1403495
U1   -0.4203174
U2    0.8490047
GDP   0.1130753
Ineq  0.4600019
Prob -0.3829873
Time -0.1661376
```

c)

Computing this 50 times with randomly generated splits:

```
# reload to be safe
crime = read.table('http://www2.stat.duke.edu/~pdh10/FCBS/Exercises/crime.dat',
             header = T)


# create train / test index
index = createDataPartition(crime$y,50)

lm_mse_output = c()
bayes_mse = c()

for (i in seq_len(length(index))){

  # train test split
  train = crime[index[[i]],]
```

```
  test = crime[-index[[i]],]

  # fit lm and calculate mse
  compare_lm = lm(y ~ ., train)
  lm_y_hat = predict(compare_lm,test[,-1])
  mse_lm = mean((test$y - lm_y_hat)^2)

  #output lm mse
  lm_mse_output = c(lm_mse_output,mse_lm)

  # bayesian linear regression
  bayes_y_hat = as.matrix(test[,-1]) %*% bayes_regression(train)
  bayes_mse_iter = mean((test$y - bayes_y_hat)^2)
  bayes_mse = c(bayes_mse, bayes_mse_iter)
}

comparison = data.frame(lm_mse = lm_mse_output,
           bayes_mse = bayes_mse)

plot(comparison$lm_mse,comparison$bayes_mse)
```



A view of the data for comparison:

```
comparison
```

```
    lm_mse bayes_mse
```

1	0.8146906	0.6956671
2	0.7738925	0.7115649
3	0.5684839	0.5193631
4	2.1298150	1.0970809
5	1.3935775	1.2439232
6	1.8594369	1.5687375
7	0.6231743	0.5092549
8	0.8616642	0.8064846
9	1.6313963	1.5701028
10	0.4725060	0.4582208
11	1.3476029	0.8734647
12	1.0904493	0.6066986
13	1.0230536	0.9696680
14	1.2045497	1.0903973
15	1.0092768	0.7541644
16	0.7958123	0.6435651
17	1.0870427	0.8795259
18	0.6642440	0.5796438
19	0.5062274	0.4684977
20	1.0163652	0.9464179
21	0.6181873	0.6131365
22	0.8597465	0.8362759
23	1.7264788	0.5518417
24	3.2497427	2.3150086
25	0.9260965	0.5975717
26	1.0562507	0.8538619
27	0.9039988	0.7315319
28	1.1359765	0.8257626
29	1.1133121	1.0482770
30	0.5832191	0.5278355
31	0.6830123	0.6352482
32	2.1665810	1.3080855
33	0.6578121	0.6412696
34	1.4850009	1.2800161
35	0.6736884	0.6282452
36	2.6196529	1.4791580
37	1.0080959	0.8699658
38	1.4090367	1.3540238
39	2.2642459	1.5617315
40	0.8999472	0.5892338
41	1.0635055	0.8324255
42	0.5848024	0.5649721
43	1.1810910	0.8775088

```
44 1.5944559 1.4322573
45 0.7016817 0.6990616
46 0.4619074 0.4439926
47 1.2716538 0.9274438
48 0.9326815 0.4593255
49 0.7943118 0.7492219
50 1.0498071 0.6326209
```