SNP-wise Selection Preliminary Algorithm

Stuart Brabbs

We now want to modify the previous log Bayes Factor function to calculate the Bayes Factor for a model with multiple SNPs. We will then want to test including this weighting in the previous SNP-wise algorithm, and see how it performs in the "three-body problem" we previously explored with SuSiE.

We first need to implement the logBF function but for multiple SNPs in the model. This requires us to modify the design matrix $\mathbf{X}$ into a $(n \text{ x } (2p+1))$ matrix with a first column of all 1's, and then two columns per SNP, with one indicating the number of minor alleles (0, 1, or 2) and the next indicating heterozygosity (0 if no, 1 if yes). We also must modify the matrix ν to be a $((2p+1) \text{ x } (2p+1))$ diagonal matrix with the first element σ_μ^2, followed by $2p$ pairs of σ_a^2 followed by σ_d^2 along the diagonal. It also requires that we multiply the $log_{10}\sigma_a^2$ and $log_{10}\sigma_d^2$ terms in the output by p. To make this work, we also remove the complete data filter (need to reformulate to filter matrices instead):

```
logBF = function(g,y,sigmaa,sigmad,sigmamu) {
  if (is.null(dim(g)[2])){
  subset = complete.cases(y) & complete.cases(g)
  y=y[subset]
  g=g[subset]
  n=length(g)
  X = cbind(rep(1,n), g, g==1)
  invnu = diag(c(0,1/sigmaa^2,1/sigmad^2))
  invOmega = invnu + t(X) %*% X
  B = solve(invOmega, t(X) %*% cbind(y))
  invOmega0 = n
  return(-0.5*log10(det(invOmega)) + 0.5*log10(invOmega0) - log10(sigmaa) -   log10(sigm
ad) - (n/2)*(log10(t(y- X %*% B) %*% y) - log10(t(y) %*% y - n*mean(y)^2)))
  }
  else {
  p=dim(g)[2]
  n=dim(g)[1]
  X = rep(1,n)
  for (i in c(1:p)) {
    X = cbind(X, g[,i], g[,i]==1)
  }
  invnu = diag(c(1/sigmamu^2,rep(c(1/sigmaa^2,1/sigmad^2), p)))
  invOmega = invnu + t(X) %*% X
  B = solve(invOmega, t(X) %*% cbind(y))
  invOmega0 = n
  return(-0.5*log10(det(invOmega)) + 0.5*log10(invOmega0) - p*log10(sigmaa) -   p*log10
(sigmad) - (n/2)*(log10(t(y- X %*% B) %*% y) - log10(t(y) %*% y - n*mean(y)^2)))
  }
}
```

We now have our logBF function for any combinatin of SNPs in a model. We can now work out a potential algorithm incorporating both the weight function and the SNP-wise regression:

```r
SNPwise = function(X, y, sigmaa, sigmad, sigmamu){
  df <- data.frame(X, y)
  snps <- dim(X)[2]
  bfs <- c()
  snpprobs <- rep(0, snps)
  for (i in c(1:snps)){
    colnames(df)[i] <- paste0("x", i)
  }
  for (i in c(1:snps)){
    reg <- stepwise(df, y = colnames(df)[snps + 1], include = colnames(df)[i], selection
= "forward")
    selected <- reg$variate[-1]
    selmatrix <- data.matrix(df[,selected])
    newbf <- logBF(selmatrix, y, sigmaa, sigmad, sigmamu)
    for (i in c(1:snps)){
      if (colnames(df)[i] %in% selected) {
        snpprobs[i] <- snpprobs[i] + newbf
      }
    }
    bfs <- c(bfs, newbf)
  }
  normalizer <- sum(bfs)
  snpprobs <- snpprobs/normalizer
  return(snpprobs)
}
```

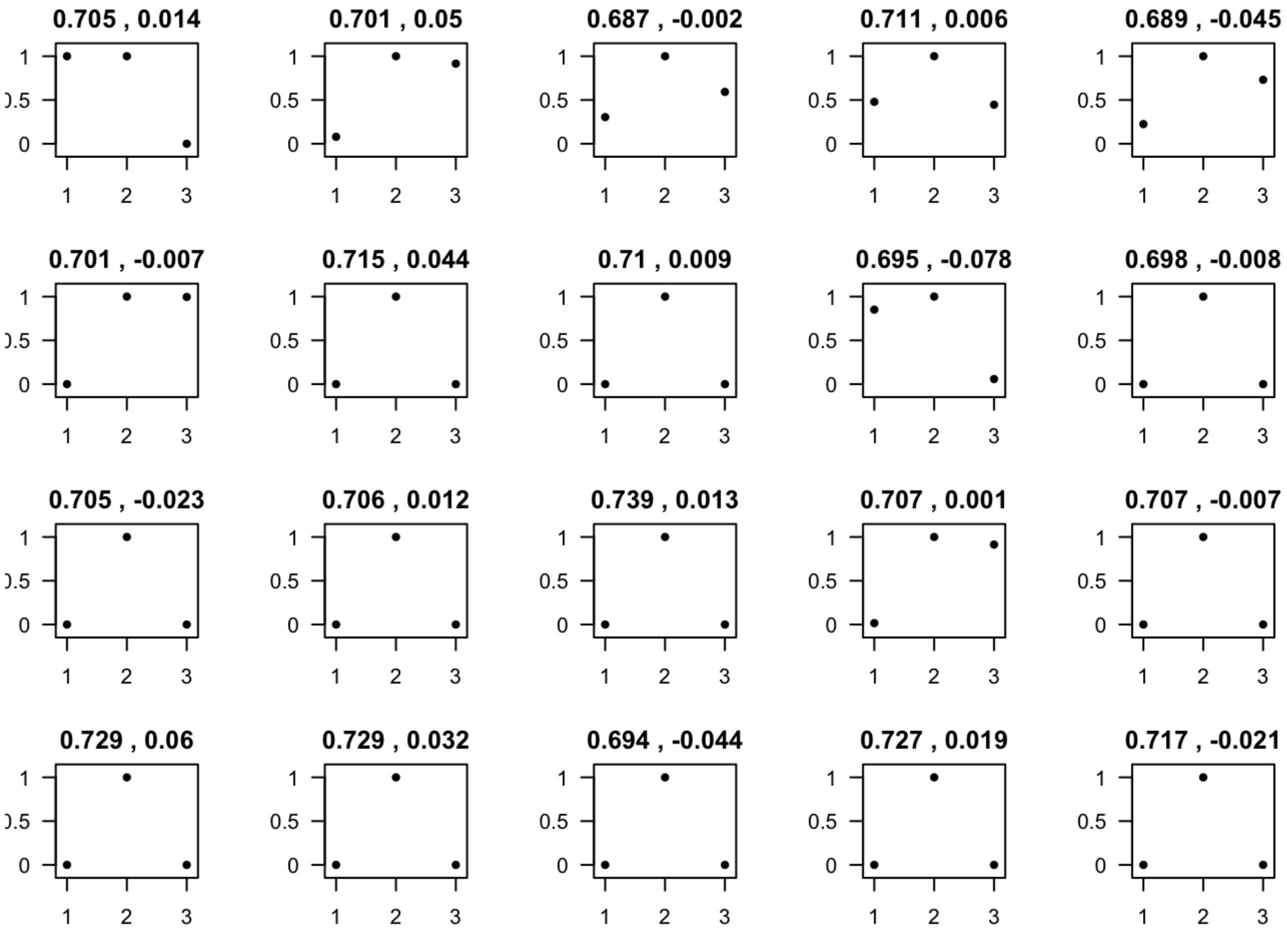
Now that we have defined the preliminary algorithm, we can try applying it the "three-body problem". We first reproduce the results from using SuSiE for comparison:

```r
set.seed(100)
sds <- sort(seq(0.02, 0.4, by=0.02), decreasing = TRUE)
cors <- cbind(rep(0,20), rep(0,20))
par(mfrow = c(4,5))
for (i in c(1:20)) {
  x1 <- rnorm(1000)
  x3 <- rnorm(1000)
  x2 <- x1 + x3 + rnorm(1000, sd = sds[i])
  X <- cbind(x1, x2, x3)
  y <- x1 + x3 + rnorm(1000)

  pcor1 <- round(max(cor(x1, x2), cor(x2,x3)), digits = 3)
  pcor2 <- round(cor(x1,x3), digits = 3)
  cors[i,1] <- pcor1
  cors[i,2] <- pcor2

  res <- susie(X, y, L= 2)

  par(mar = c(3,2,2,3))
  title <- paste(pcor1, pcor2, sep = " , ")
  plot(res$pip, xlab = "Predictor", ylab = "PIP", main = title, pch = 20, xlim = c(0.9,
3.1), ylim = c(-0.1, 1.1), xaxt = "n", yaxt = "n")
  axis(side = 2, at = c(0,0.5,1), labels = c("0", "0.5", "1"), las = 1)
  axis(side = 1, at = c(1,2,3), labels = c("1", "2", "3"))
}
```



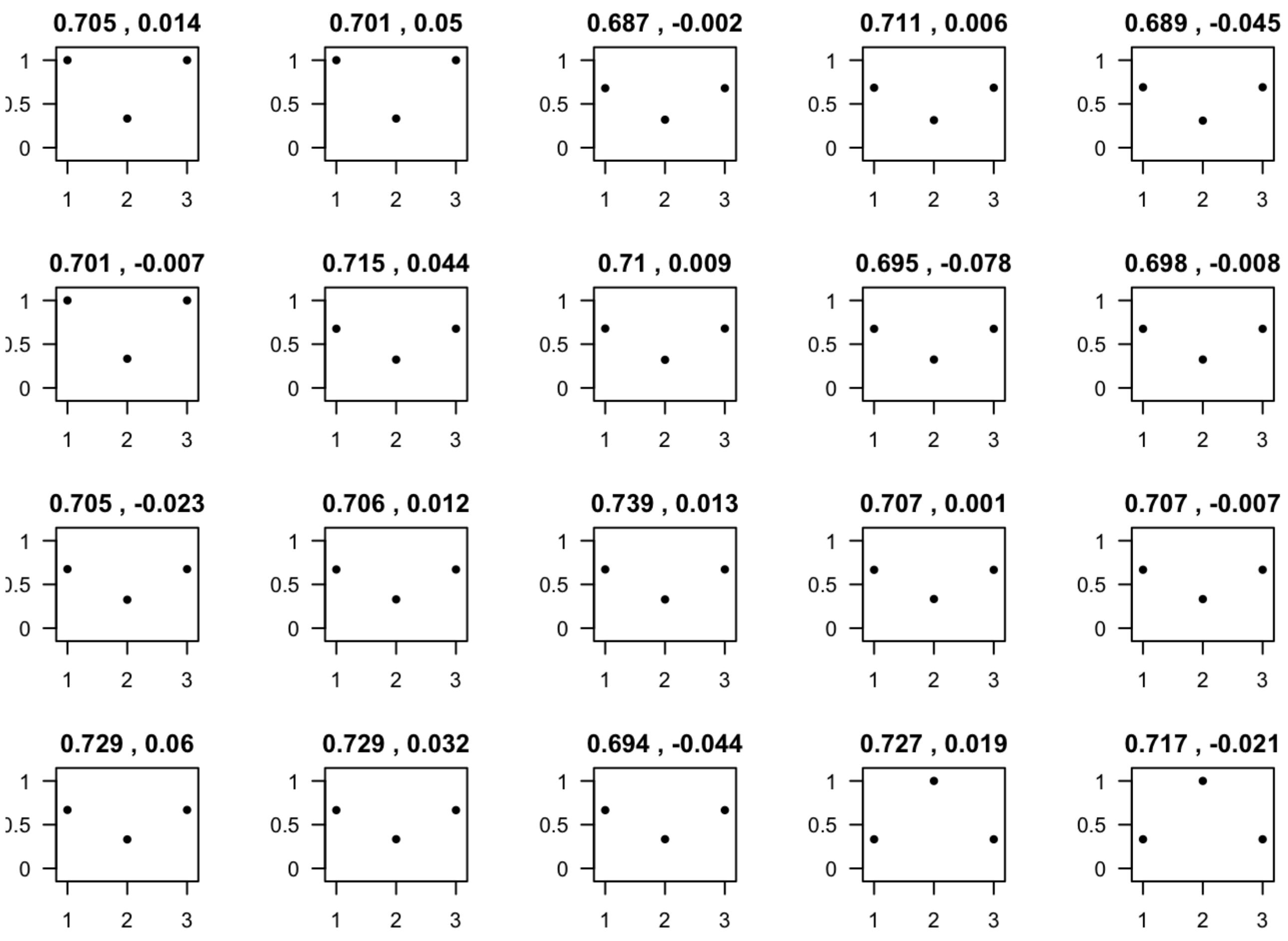
We then use the same data, but apply it to the new algorithm and plot the resulting "inclusion scores":

```r
set.seed(100)
sds <- sort(seq(0.02, 0.4, by=0.02), decreasing = TRUE)
cors <- cbind(rep(0,20), rep(0,20))
par(mfrow = c(4,5))
for (i in c(1:20)) {
  x1 <- rnorm(1000)
  x3 <- rnorm(1000)
  x2 <- x1 + x3 + rnorm(1000, sd = sds[i])
  X <- cbind(x1, x2, x3)
  y <- x1 + x3 + rnorm(1000)

  pcor1 <- round(max(cor(x1, x2), cor(x2,x3)), digits = 3)
  pcor2 <- round(cor(x1,x3), digits = 3)
  cors[i,1] <- pcor1
  cors[i,2] <- pcor2

  res <- SNPwise(X, y, 0.5, 0.5, 100)

  par(mar = c(3,2,2,3))
  title <- paste(pcor1, pcor2, sep = " , ")
  plot(res, xlab = "Predictor", ylab = "Inclusion Score", main = title, pch = 20, xlim =
c(0.9,3.1), ylim = c(-0.1, 1.1), xaxt = "n", yaxt = "n")
  axis(side = 2, at = c(0,0.5,1), labels = c("0", "0.5", "1"), las = 1)
  axis(side = 1, at = c(1,2,3), labels = c("1", "2", "3"))
}
```



From the plots, we can see that the new algorithm never assigns a variable exactly 0 (as expected), but also begins to spread the "score" amongst the variables much earlier than SuSiE. However, while SuSiE misidentifies the model immediately, the new algorithm assigns the true causal variables x1 and x3 significantly higher score than x2 up until the smallest standard deviations in noise in x2. This comes with the caveat that these two are rarely at 100%, but generally are above 60%.