Practical 4 Bayesian methods for genomic prediction

Winter School 2022 Module 4 Quantitative Genetics

2022-06-23

In this practical you will perform genomic prediction using Bayesian methods. We will use the same data sets in the BLUP session, i.e., a small data set with 10 SNPs and simulated phenotypes for 325 bulls , and a larger data set of 10,000 GWAS individuals and 277,719 SNPs. We will use R script and GCTB to do these analyses.

Analysis of the small data set using R script

Read the data in R.

```
nmarkers <- 10       #number of markers
nrecords <- 325      #number of records

x <- matrix(scan("/data/module4/prac4/xmat.inp"),ncol=nmarkers,byrow=TRUE)
y <- matrix(scan("/data/module4/prac4/yvec.inp"),byrow=TRUE)
x_prog <- matrix(scan("/data/module4/prac4/xmat_prog.inp"),ncol=nmarkers,byrow=TRUE)
y_prog <- matrix(scan("/data/module4/prac4/yvec_prog.inp"),byrow=TRUE)
```

SNP-BLUP as benchmark

The code for running BLUP is shown here. You can run BLUP and use the prediction accuracy from BLUP as benchmark to compare those from the Bayesian methods below.

```
# set value for lambda
lambda    <- 10

# need a vector of ones and a identity matrix with the size of the number of SNPs
ones <- array(1,c(nrecords))
ident_mat <-diag(nmarkers)

# build the left hand side of the mixed model equations
coeff <- array(0,c(nmarkers+1,nmarkers+1))
coeff[1:1, 1:1] <- t(ones)%*%ones
coeff[1:1,2:(nmarkers+1)] <- t(ones)%*%x
coeff[2:(nmarkers+1), 2:(nmarkers+1)] <- t(x)%*%x + ident_mat*lambda
coeff[2:(nmarkers+1), 1:1] <- t(coeff[1:1,2:(nmarkers+1)])

# build the right hand side of the mixed model equations
rhs <- array(0, c(nmarkers+1,1))
rhs[1,1]=t(ones)%*%y
rhs[2:(nmarkers+1),1]=t(x)%*%y

# get BLUP solution
solution_vec <- solve(coeff,rhs)
```

```
# get the predicted genetic value (GEBV)
ghat_blup=x_prog%*%solution_vec[-1]

# prediction R-square
summary(lm(y_prog~ghat_blup))$r.squared
```

```
## [1] 0.5049367
```

A Bayesian model with point-normal prior (BayesCπ)

For the first exercise, we will analyse this small data set using BayesCπ.

$$\beta_j \begin{cases} \sim N(0, \sigma^2_\beta) & \text{with probability } \pi \\ = 0 & \text{with probability } 1-\pi \end{cases}$$

where parameter π is estimated from the data. It can be seen that when $\pi = 0$, the prior collapses to a normal distribution, i.e., $\beta_j \sim N(0, \sigma^2_\beta)$, which is the assumption in BLUP.

We will analyse the data with a script written in R language, `BayesR.R`. Let us load it in R.

```
source("/data/module4/prac4/bayesr.R")
```

The script includes a function called `bayesr`. BayesR is a general model where each SNP effects is assumed to follow a mixture distribution of normal distributions. BayesCπ is a special case of BayesR. The input parameters are

```
# Input parameters for bayesr. Do not run this.
bayesr = function(X, y, niter, gamma, startPi, startH2)
```

where `niter` is the number of iterations for MCMC sampling, `X` is the genotype matrix, `y` is the vector of phenotypes, `gamma` is a vector of scaling factor for the variance of a mixture distribution, `startPi` is a vector of starting values for the π parameter, `startH2` is the starting value of SNP-based heritability. Note that the number of elements in `gamma` and `startPi` define the number of mixture components, and therefore need to be matched.

We can use this program to run BayesCπ by specifying the mixture model as

```
gamma = c(0, 1)
startPi = c(0.8, 0.2)
```

This means the SNP effects are assumed to be normally distributed with a probability of 0.2 and have a point mass at zero with a probability of 0.8. Now we can run the analysis using command

```
bayesc.res = bayesr(X = x, y = y, gamma = gamma, startPi = startPi)
```

```
##
##  iter  100, nnz =    2, sigmaSq =  1.171, h2 =  0.406, vare =  3.181, varg =  2.177
##
##  iter  200, nnz =    4, sigmaSq =  0.531, h2 =  0.429, vare =  3.172, varg =  2.380
##
##  iter  300, nnz =    4, sigmaSq =  1.036, h2 =  0.502, vare =  2.851, varg =  2.874
##
##  iter  400, nnz =    9, sigmaSq =  0.448, h2 =  0.440, vare =  2.985, varg =  2.350
##
##  iter  500, nnz =    3, sigmaSq =  0.681, h2 =  0.373, vare =  3.282, varg =  1.949
##
##  iter  600, nnz =    4, sigmaSq =  1.379, h2 =  0.469, vare =  2.748, varg =  2.429
```

```
##
##  iter  700, nnz =    4, sigmaSq =  0.634, h2 =  0.393, vare =  3.345, varg =  2.167
##
##  iter  800, nnz =    5, sigmaSq =  0.599, h2 =  0.422, vare =  2.835, varg =  2.071
##
##  iter  900, nnz =    4, sigmaSq =  1.716, h2 =  0.476, vare =  2.511, varg =  2.278
##
##  iter 1000, nnz =    7, sigmaSq =  0.446, h2 =  0.450, vare =  2.799, varg =  2.291
##
## Posterior mean:
##       Pi1       Pi2       Nnz   SigmaSq        h2      Vare      Varg
## 0.4827754 0.5172246 5.1970000 1.0861824 0.4543008 2.8317909 2.3654357
```

After the script has run, you can find the sampled values for the model parameters and SNP effects for each iteration in the result list. For example, you can look at the posterior mean and standard deviation for each parameter by

```
colMeans(bayesc.res$par)
```

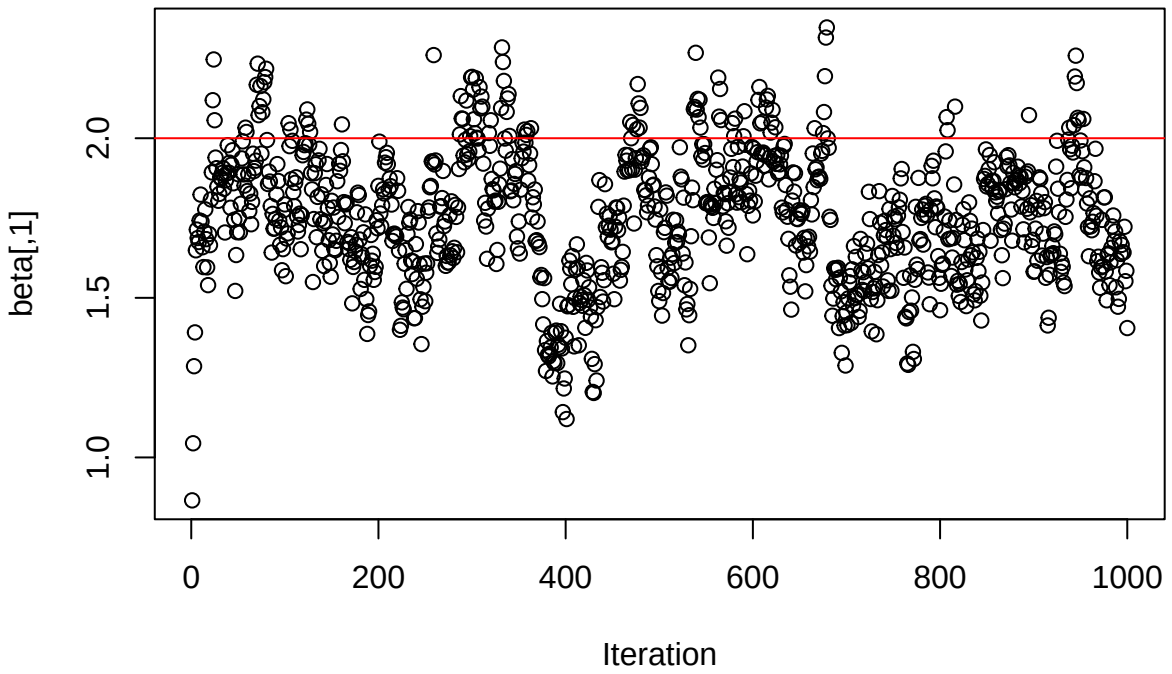
```
##       Pi1       Pi2       Nnz   SigmaSq        h2      Vare      Varg
## 0.4827754 0.5172246 5.1970000 1.0861824 0.4543008 2.8317909 2.3654357
```

```
apply(bayesc.res$par, 2, sd)
```

```
##        Pi1        Pi2        Nnz    SigmaSq         h2       Vare       Varg
## 0.23031299 0.23031299 2.28810542 0.86292603 0.03698011 0.22289594 0.28890306
```

You can use the plotting facilities in R to investigate changes in the parameters over iterations. For example, to look at the effect of the first marker across iterations, you would enter the command

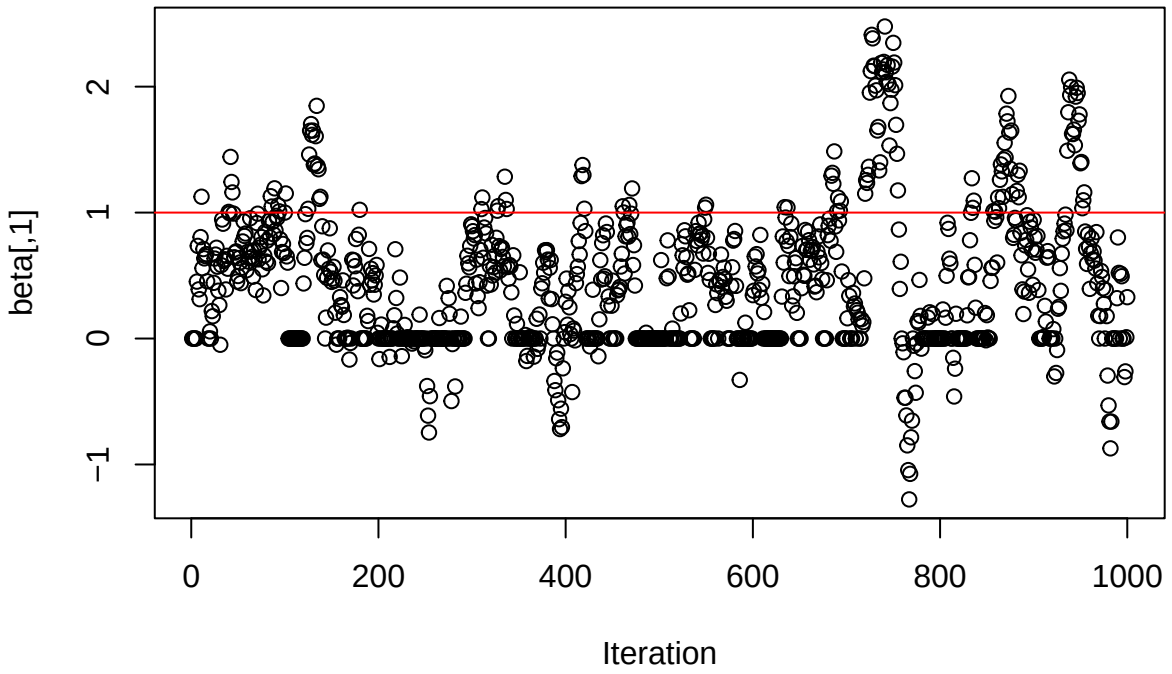
```
#png("beta1_vs_iter.png")
plot(1:nrow(bayesc.res$beta), bayesc.res$beta[,1], xlab="Iteration", ylab="beta[,1]")
abline(h = 2, col="red")
```




```
#dev.off()
```

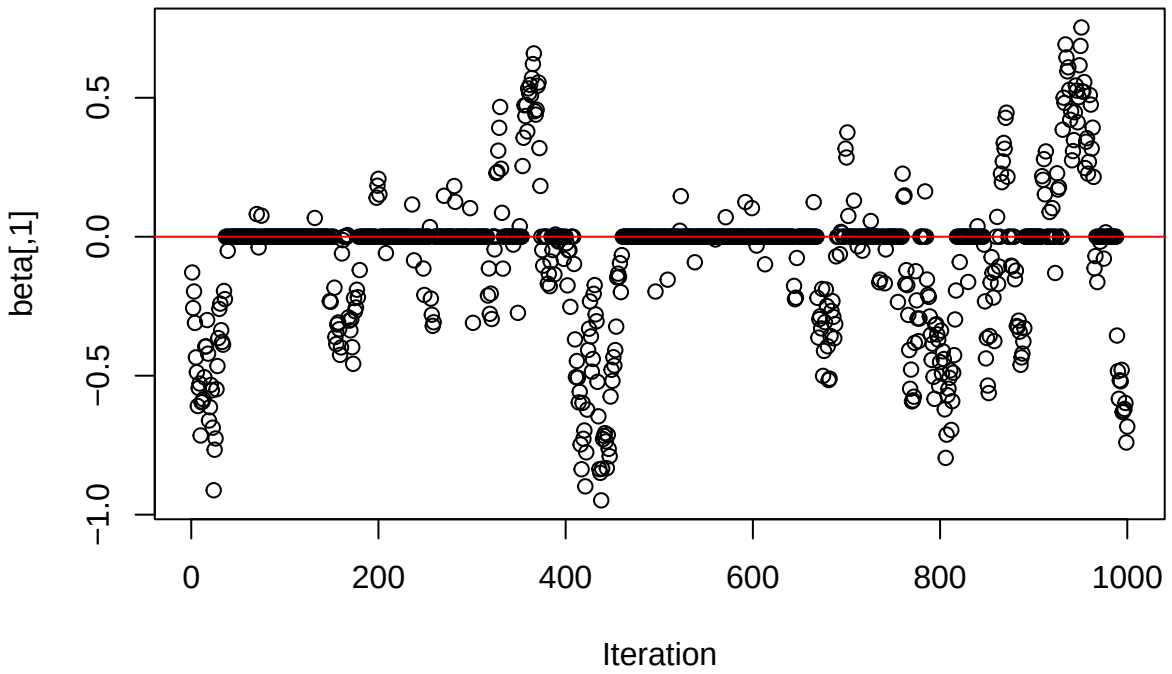
Question 1: Use this command to investigate each of the parameters in turn. Do they appear to be fluctuating about the correct values?

```
#png("beta2_vs_iter.png")
plot(1:nrow(bayesc.res$beta), bayesc.res$beta[,2], xlab="Iteration", ylab="beta[,1]")
abline(h = 1, col="red")
```




```
#dev.off()
```

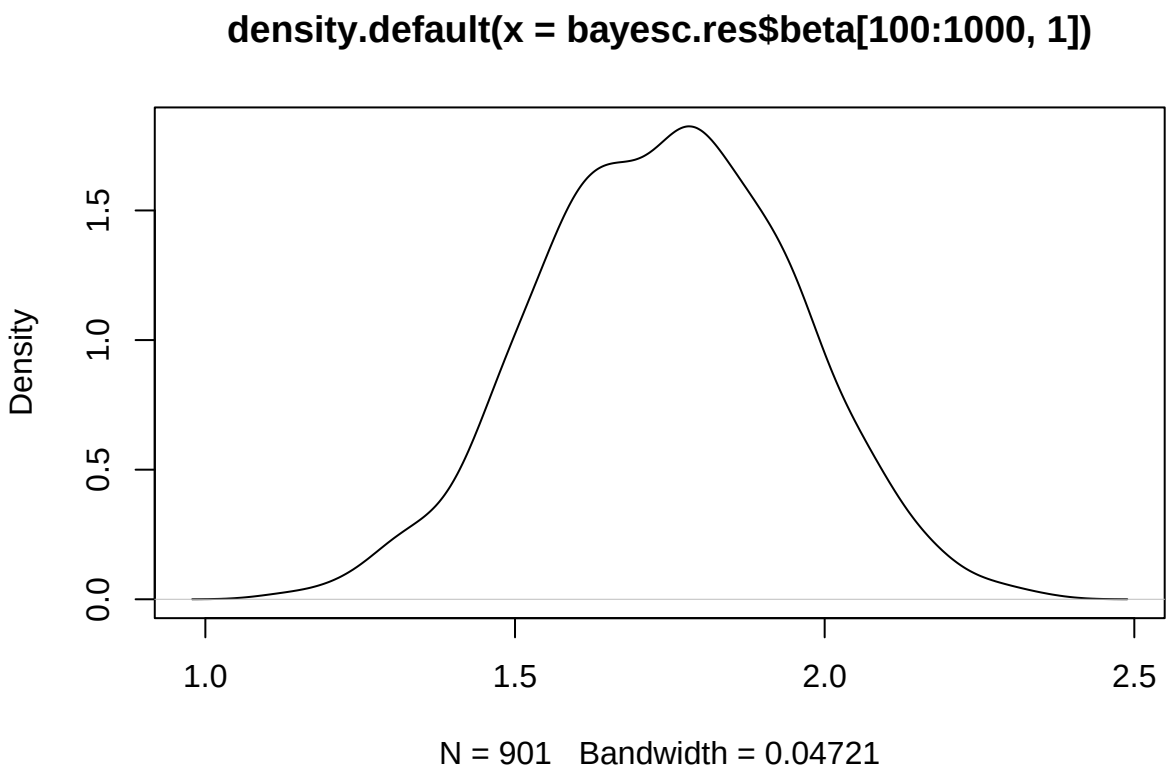
```
#png("beta5_vs_iter.png")
plot(1:nrow(bayesc.res$beta), bayesc.res$beta[,5], xlab="Iteration", ylab="beta[,1]")
abline(h = 0, col="red")
```



```
#dev.off()
```

We can also plot the posterior distribution, for example for the effect of the third SNP. We would discard the first 100 iterations of the program as "burn in":

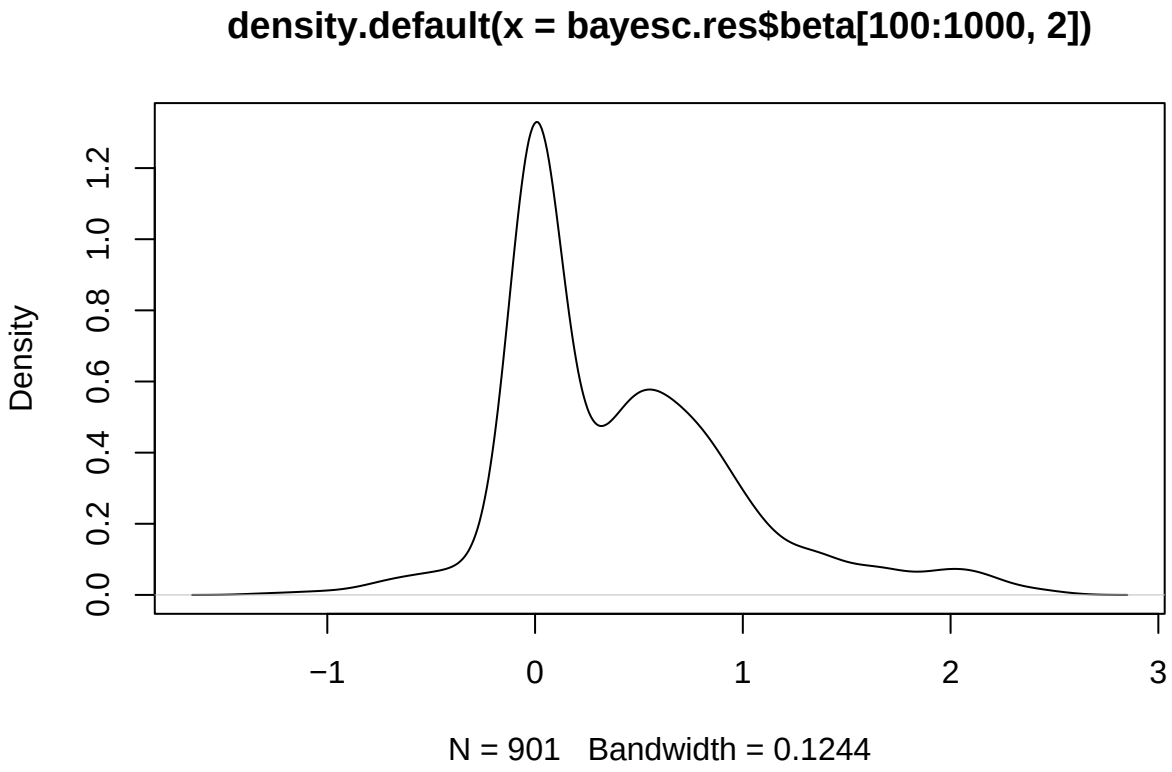
```
#png("dist_beta1.png")
plot(density(bayesc.res$beta[100:1000,1]))
```




```
#dev.off()
```

Question 2: Does the distribution appear to be normal? What about the distributions of the other parameters?

```
#png("dist_beta2.png")
plot(density(bayesc.res$beta[100:1000,2]))
```




```
#dev.off()
```

To get the SNP effect estimates, we would calculate the posterior mean of SNP effects:

```
betaMean = colMeans(bayesc.res$beta)
```

Question 3: What's the prediction accuracy using BayesCπ? How does it compare to that from BLUP? What causes the difference?

```
ghat_bayesc=x_prog %*% betaMean
summary(lm(y_prog ~ ghat_bayesc))$r.squared
```

```
## [1] 0.5524594
```

Bayesian approach using multi-component mixture prior (BayesR)

BayesR prior assumes that some SNPs have zero effect, some have small effects, and some have large effects, by assuming a mixture of multiple normal distributions, including a point mass at zero.

$$\beta_j \sim \sum_k \pi_k N(0, \gamma_k \sigma_\beta^2)$$

For example, we can set a 4-component mixture model:

```
gamma = c(0, 0.01, 0.1, 1)
startPi = c(0.5, 0.3, 0.15, 0.05)
bayesr.res = bayesr(X = x, y = y, gamma = gamma, startPi = startPi)
```

```
##
##  iter  100, nnz =    6, sigmaSq =  1.806, h2 =  0.430, vare =  3.498, varg =  2.639
##
##  iter  200, nnz =    7, sigmaSq =  0.979, h2 =  0.387, vare =  3.099, varg =  1.953
```

```
##
##  iter  300, nnz =    7, sigmaSq =  1.453, h2 =  0.497, vare =  2.657, varg =  2.622
##
##  iter  400, nnz =    9, sigmaSq =  2.859, h2 =  0.460, vare =  2.919, varg =  2.490
##
##  iter  500, nnz =    4, sigmaSq =  0.954, h2 =  0.442, vare =  2.843, varg =  2.255
##
##  iter  600, nnz =    8, sigmaSq =  3.609, h2 =  0.448, vare =  2.817, varg =  2.290
##
##  iter  700, nnz =    3, sigmaSq =  2.199, h2 =  0.505, vare =  2.655, varg =  2.708
##
##  iter  800, nnz =   10, sigmaSq =  1.540, h2 =  0.476, vare =  2.956, varg =  2.683
##
##  iter  900, nnz =    9, sigmaSq =  2.487, h2 =  0.434, vare =  2.767, varg =  2.123
##
##  iter 1000, nnz =    4, sigmaSq =  4.430, h2 =  0.429, vare =  2.788, varg =  2.096
##
## Posterior mean:
##       Pi1       Pi2       Pi3       Pi4       Nnz   SigmaSq        h2      Vare
## 0.2409412 0.2631374 0.2468867 0.2490347 7.6720000 2.5676517 0.4594356 2.8126619
##      Varg
## 2.3978588
```

Question 4: Is the prediction accuracy further improved? Why or why not?

```
betaMean = colMeans(bayesr.res$beta)
ghat_bayesr=x_prog %*% betaMean
summary(lm(y_prog ~ ghat_bayesr))$r.squared
```

```
## [1] 0.5466322
```

You can also play around with the number of mixture components and see how sensitive the results are.

Analysis of a larger data set using GCTB

GCTB (https://cnsgenomics.com/software/gctb/#Overview) is a c++ software for performing Bayesian analysis of large-scale genomic data. It takes the PLINK bed file as input and generate estimates for model parameters and SNP effects for polygenic prediction. You can use the following code to run BayesR analysis. However, it will take about 10 mins to finish the genome-wide analysis. You can add `--chr 1` to just run the analysis using SNPs on chromosome 1, which should take about 1 min to finish.

```
bfile="/data/module4/prac4/gwas"
pheno="/data/module4/prac4/simu.phen"
covar="/data/module4/prac4/covariates.cov"
gctb --bayes R \
     --bfile $bfile \
     --pheno $pheno \
     --covar $covar \
     --original-model \
     --chain-length 1000 \
     --burn-in 100 \
     --out bayesr
```

Result file `bayesr.parRes` shows the estimates of model parameters. Result file `bayesr.snpRes` shows the estimates of SNP effects. We have generated the results from the genome-wide analysis, which can be found at

```
cat /data/module4/prac4/bayesr.parRes
```

```
## Posterior statistics from MCMC samples:
##
##               Mean            SD
##    NumSnp1    275748.812500   555.326721
##    NumSnp2    1152.233276     620.240845
##    NumSnp3    808.711121      82.797646
##    NumSnp4    9.288889        6.821933
##        Vg1    0.000000        0.000000
##        Vg2    0.112438        0.060669
##        Vg3    0.841062        0.069600
##        Vg4    0.046500        0.032679
##    SigmaSq    10.039310       0.587386
##     ResVar    957.568726      47.679611
##     GenVar    977.727234      50.605698
##        hsq    0.505170        0.024771
```

`NumSnp1-NumSnp4` are the numbers of SNPs in the mixture components 1 to 4 (component 1: zero effect; component 2: small effects that explain 0.01% heritability; component 3: medium effects that explain 0.1% heritability; component 4: large effects that explain 1% heritability). `Vg1-Vg4` are the proportions of variance explained by the SNPs in each component. `hsq` is the SNP-based heritability estimate. We set chain length to be 1000 here for demonstration purpose. In practice, we recommend to run at least 5,000 iterations. It is common to discard the first 20% of the samples as burn in (i.e., 1,000 if chain length is set to be 5,000).

Question 5: Are the estimates from the genome-wide BayesR analysis consistent with the true values in simulation? Note that we simulated 1,000 causal variants with a trait heritability of 0.5.

Let's also have a look at the SNP effect results:

```
head /data/module4/prac4/bayesr.snpRes
```

```
##     Id                Name  Chrom     Position     A1     A2        A1Frq     A1Effect          SE
##      1          rs12562034      1       768448      1      2     0.102253    -0.004296    0.040525
##      2           rs4040617      1       779322      2      1     0.128397     0.000000    0.000000
##      3           rs4970383      1       838555      1      2     0.247147    -0.000746    0.007038
##      4            rs950122      1       846864      1      2     0.197399     0.010847    0.102333
##      5           rs6657440      1       850780      2      1     0.392077     0.000000    0.000000
##      6          rs13303101      1       862124      1      2     0.019920     0.012047    0.113647
##      7           rs1110052      1       873558      2      1     0.274854     0.000000    0.000000
##      8           rs3748592      1       880238      1      2     0.052766     0.000000    0.000000
##      9           rs3748593      1       880390      1      2     0.026616    -0.001978    0.018656
```

We then use PLINK to perform prediction:

```
target="/data/module4/prac4/target"
snpRes="/data/module4/prac4/bayesr.snpRes"
plink --bfile $target --score $snpRes 2 5 8 header sum center --out simu.bayesr
```

Use R to evaluate the prediction accuracy

```
phenFile="/data/module4/prac4/simu.phen"
covFile="/data/module4/prac4/covariates.cov"
indlistFile="/data/module4/prac4/target.indlist"
prsFile="simu.bayesr.profile"
Rscript /data/module4/prac4/get_pred_r2.R $phenFile $covFile $indlistFile $prsFile
```

Question 6: How does the prediction accuracy from BayesR compare to that from C+PT?