bayesGen: Association analysis of genomic data using a Bayesian shared component model

Juan J Abellán, Carlos Abellán, Juan R González*

July 14, 2011

Joint Research Unit on Genomics and Health, Centre for Public Health Research (CSISP) and Cavanilles Institute for Biodiversity and Evolutionary Biology, University of Valencia, Valencia, Spain
CIBER Epidemiología y Salud Pública (CIBERESP)
Center for Research in Environmental Epidemiology (CREAL), Barcelona, Spain

*jrgonzalez@creal.cat
http://www.creal.cat/jrgonzalez/software.htm

Contents

1 Introduction

This document provides an overview of the `bayesGen` package that is available at CRAN (http://cran.r-project.org/). The package implements a Bayesian approach for genetic association studies. We propose a shared-component model to tease out the genotype information that is common to cases and controls from the one that is specific to cases. This allows to detect the SNPs (`bayesSNPassoc` function) or CNVs (`bayesCNVassoc` function) that show the strongest association with the disease. The model can nevertheless be applied to more than one disease. More detailed information about the model and assumptions are given in [1] and [3] in the case of analyzing SNPs or CNVs, respectively. We illustrate how to analyze SNP data by using a synthetic data set (WTCCC will also be included). The simulated data set contains information about 72 SNPs in 5 genes. It includes information for 800 individuals divided in 4 populations: controls and 3 type of cases (M1, M2, and M3). We simulated SNPs from gene 1 to be associated with M1 and SNPs from gene 3 to M3.

2 Getting started

The `bayesGen` package uses `JAGS`, a program for analysis of Bayesian hierarchical models using Markov Chain Monte Carlo (MCMC) simulation [5], to estimate model parameters. The current implementation of `bayesGen` is based on JAGS version 1.4.0. JAGS has an R interface `rjags` that is used by the package (`rjags version 1.0.3-13`).

We start by attaching the required libraries by typing

```
> library(rjags)

loading JAGS module
   basemod
   bugs
```

Then, the library is loaded by excuting

```
> library(bayesGen)
```

3 Analysis of SNP data

3.1 The data

Data can be imported from a text file or can be loaded using `snpMatrix` package (**to be supplied**). We provide a simulated example that can be loaded by typing

```
> data(sim.data)
```

The package requires to have the case-control status and the SNPs in to different objects:

```
> group <- sim.data$caco
> SNPs <- sim.data[, -1]
```

3.2 Model parameter estimates

The model can be run by using the function `bayesSNPassoc` by executing

```
mod<-bayesSNPassoc(group, SNPs)
Compiling model graph
   Declaring variables
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 1901

  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100%
  |**************************************************| 100%
  |**************************************************| 100%
```

This process takes about 5 minutes. To avoid waiting, we have saved the object `mod` that can be loaded as

```
> data(mod)
```

The function `bayesSNPassoc` prepares the data and then calls `rjags` to estimate model parameters. We have set up the following default arguments to be passed through `rjags` functions:

```
> args(bayesSNPassoc)
```

```
function (y, snps, annotation, chr, QC = 0.9, min.freq = 0.05,
    method = "JAGS", n.iter.burn.in = 10000, n.iter = 30000,
    thin = 50, n.chain = 2, ...)
NULL
```

Notice that other arguments realated to MCMC estimation using `JAGS` can be passed through this function. More details about them can be obtained at `http://calvin.iarc.fr/ martyn/software/jags/`.

3.3 Checking convergence

Before interpreting the simulations obtained from de a posteriori distri bution, Markov chains convergence migth be verified. This can be done by using the function `checkConvergence`. This function has an argument called `type` that defines the kind ob plot to be obtained. When `type="Markov chain"` (default value) the function calls to `plot.mcmc` from package `coda`. On the other hand, Gelman-Rubin plots are displayed. The function `checkConvergence` has another argument, `parameter`, to indicate the model parameter to be summaryzed. The default is 'alpha'. For example, Figure 1 can be obtained by executing:

```
> pdf("./figures/fig-check-alpha.pdf")
> checkConvergence(mod)
> dev.off()

null device
          1
```

Other model parameters (Figure 2) are summaryzed by changing the argument called `parameter`.

```
> pdf("./figures/fig-check-lambda.pdf")
> checkConvergence(mod, parameter = "log-lambda")
> dev.off()

null device
          1
```

Gelman-Rubin plot for alpha parameter can be obtained by typing

```
> pdf("./figures/fig-check-alpha-GR.pdf")
> checkConvergence(mod, type = "Gelman-Rubin")
> dev.off()
```

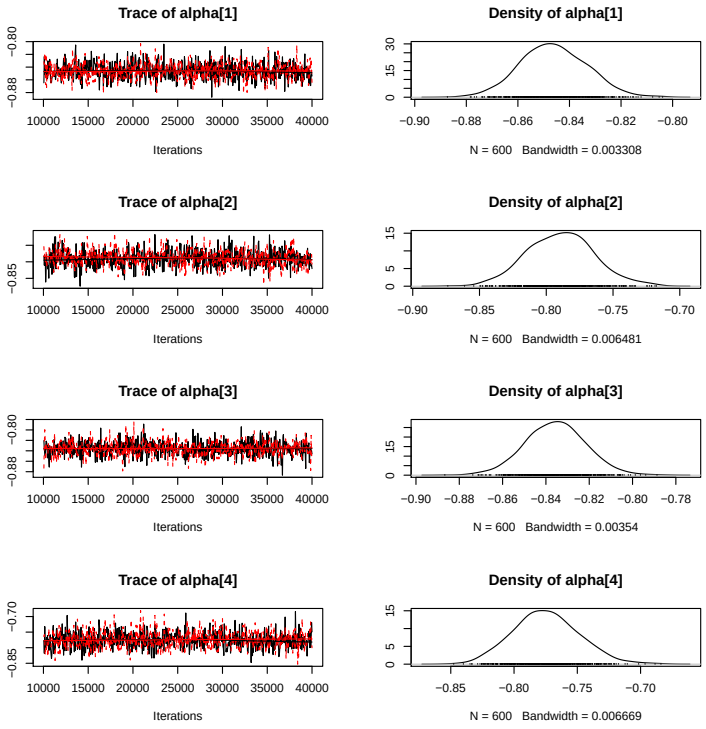

Figure 1: Check alpha ...

```
null device
          1
```

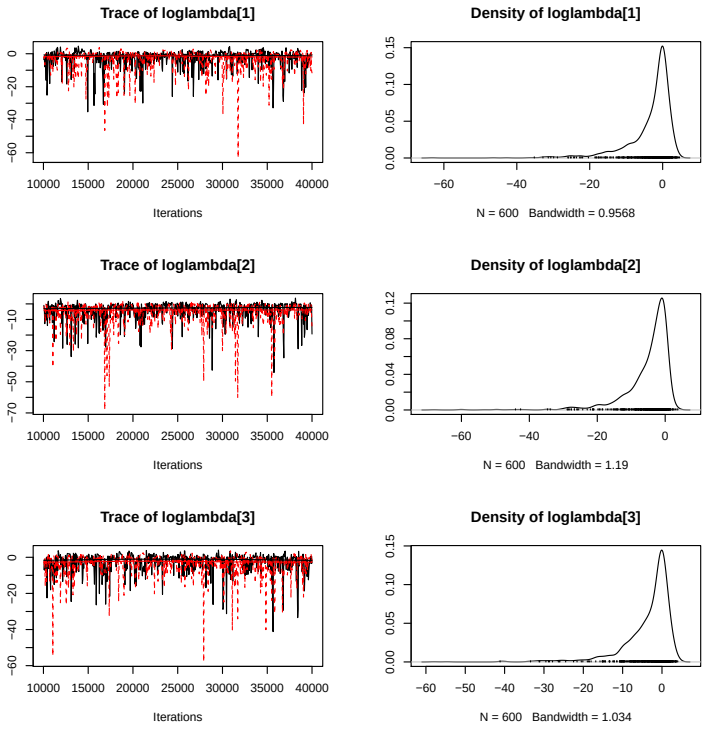

Figure 2: Check log-lambda ...

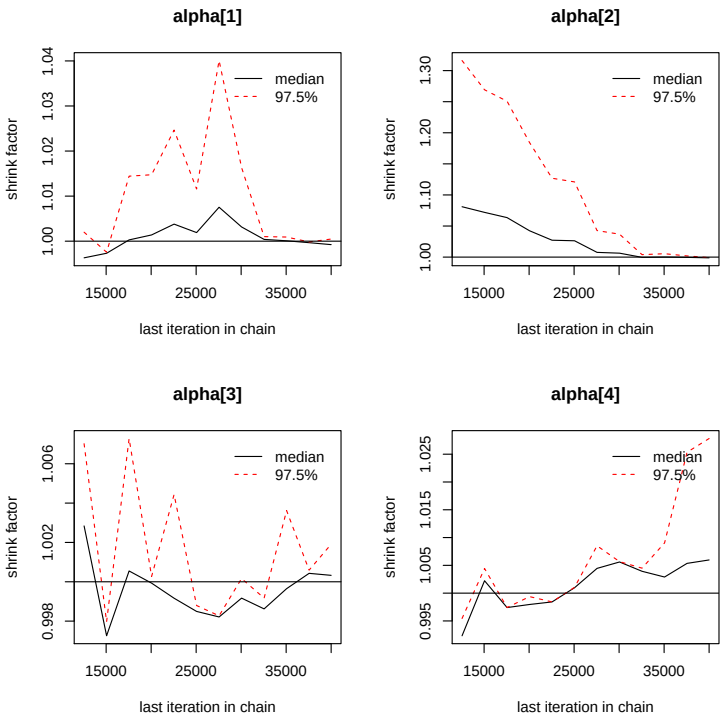

Figure 3: Check alpha ...

3.4 Results

Model parameters (intercept and shared component) can be obtained by typing:

```
> getParameters(mod)
```

```
 Intercepts (alpha):
                          2.5%        50%      97.5%
Control -0.8460544 -0.8704763 -0.8464547 -0.8205316
M1      -0.7892452 -0.8392849 -0.7891234 -0.7370540
M2      -0.8356149 -0.8632674 -0.8357018 -0.8084856
M3      -0.7745431 -0.8229356 -0.7755716 -0.7214732

 Shared component (log-lambda):
                  2.5%       50%    97.5%
M1 -3.340097 -22.49532 -1.005017 2.733238
M2 -5.131447 -25.48336 -2.869949 1.021008
M3 -3.503623 -21.05903 -1.292793 2.445820
```

On the other hand, specific and shared components can be obtained by executing

```
> pdf("./figures/fig-specific.pdf")
> plot(mod)
> dev.off()
```

```
null device
          1
```

and

```
> pdf("./figures/fig-shared.pdf")
> plot(mod, type = "shared")
> dev.off()
```

```
null device
          1
```

respectively.
Figure 4 shows the specific components for each SNP, while Figure 5 gives the shared components.
Finnally, a hierarchical clustering can be performed by using the predicted probabilities by typing:

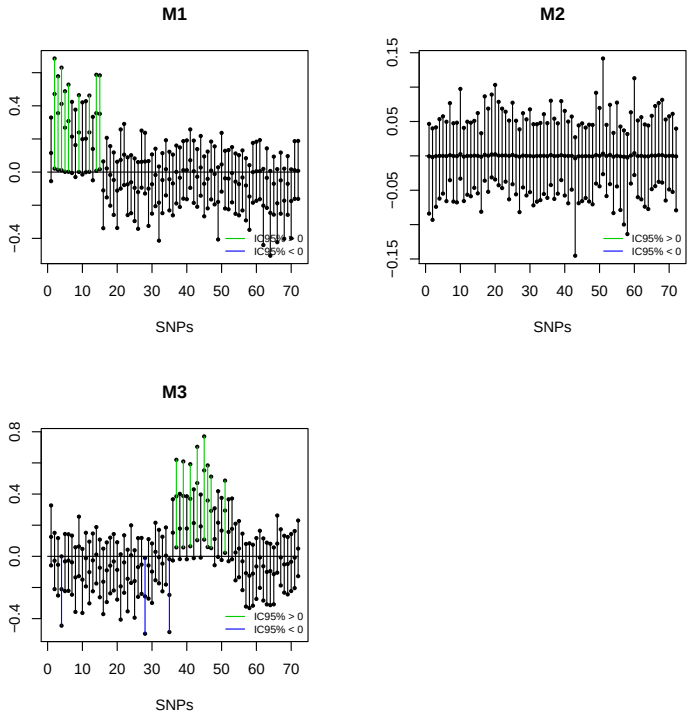

Figure 4: Specific component for the simulated data

```
> library(RColorBrewer)
> pdf("./figures/fig-heatmap.pdf")
> makeHeatmap(mod)
> dev.off()

null device
          1
```

Figure 6 shows a Heatmap were we can observe that groups M1 and M3 are different from cases and group M1 (NOTA: podria poner lo de los genes, pero hacer esto de forma general me tomara un poco de tiempo)

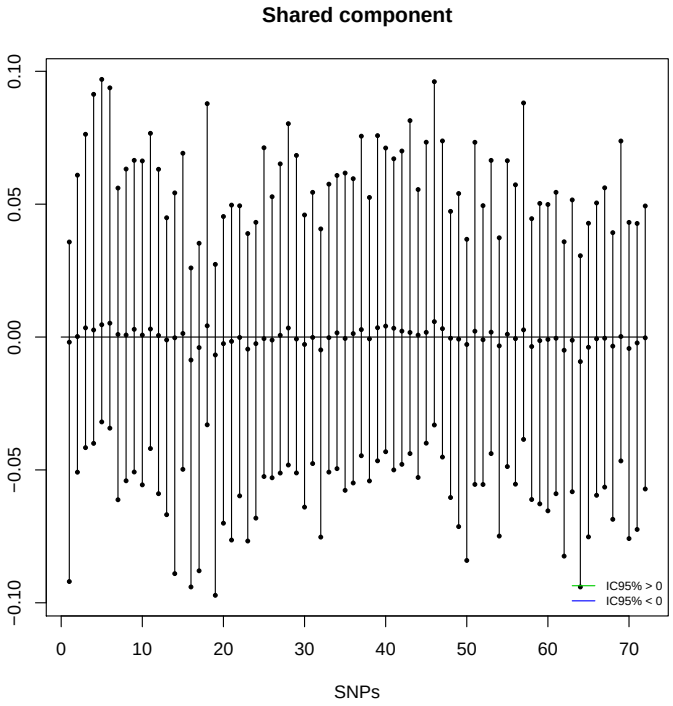

Figure 5: Shared component for the simulated data

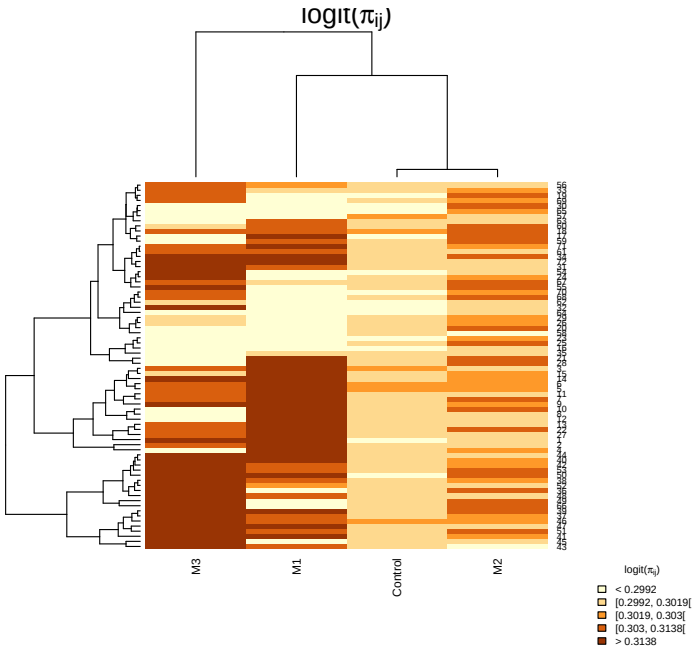

Figure 6: Heatmap for the simulated data

3.5 Model Validation

DIC (Deviance Information Criteria) is one of the most used criteria to evaluate the goodness-of-fit for a given model. We use the function `dic.samples` from `rjags` package to get such information. This deviance can be computed using two types of penalization depending on the argument `type`. The classic penalization proposed by [2] can be obtained by setting `type="pD"`, while the penalization proposed by [4] can be obtained with `type="popt"`.

```
> dic.samples(mod$model, n.iter = 2000, type = "pD")
loading JAGS module
   dic
  |**************************************************| 100%
Mean deviance:  2083
pD (Markov Error): 120.1 (0.7272)
Penalized deviance: 2203
```

4 Analysis of CNV data

Data can be imported from a text file or can be loaded using `snpMatrix` package (**to be supplied**). We provide a simulated example that can be loaded by typing

```
> data(armengol)
```

Multiple comparisons problem is address by computing confidence credible intervals at more stringent level `alpha.corrected`

```
> nCNVs <- ncol(armengol) - 1
> alpha.corrected <- (0.05/nCNVs)/2
```

Model parameter estimates are obtained using the function `bayesCNVassoc` by executing

```
mod.CNV<-bayesCNVassoc(armengol[,1], armengol[-1], method="JAGS", alpha=c(alpha.corre
```

This process takes about 6-8 minutes depending on the processor. To avoid waiting, we have saved the object `mod.CNV` that can be loaded as

```
> data(modCNV)
```

Specific components for each population can be obtained by typing:

```
> mod.CNV
```

The same information can be visually inspected in Figure 7. This figure can be obtained by executing

```
pdf("./figures/fig-specific_CNV.pdf")
plot(mod.CNV)
dev.off()
```

5 Acknowledgments

This work has been partly supported by the Spanish Ministry for Science and Innovation (MTM2008-02457) and by XXX

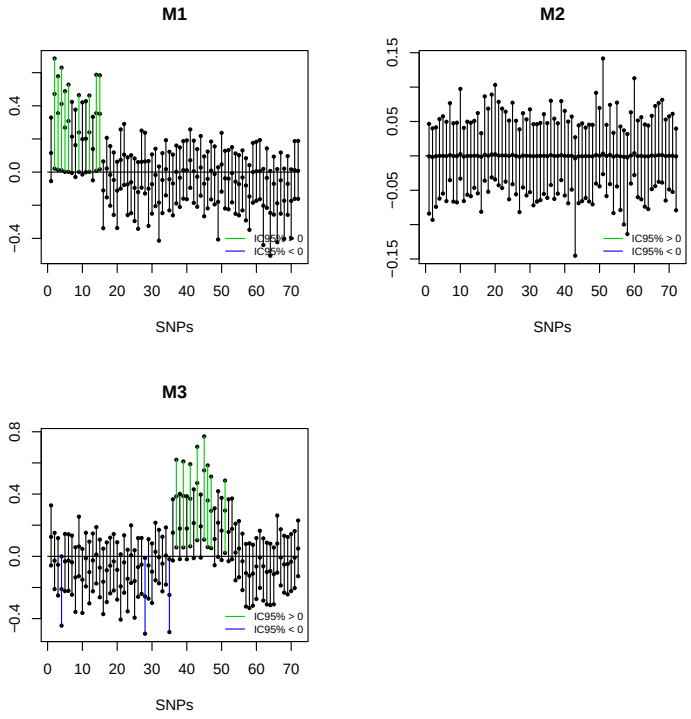

Figure 7: Specific component for `armengol` dataset

References

[1] J. J. Abellan, C Abellan, and J. R. Gonzalez. A Bayesian shared component model for genome association studies. Technical Report 1120, COBRA, 2010.

[2] D.J. DJ Spiegelhalter, N.G. Best, B.P. Carlin, and A. van der Linde. Bayesian measures of model complexity and fit (with discussion). *Journal of the Royal Statistical Societey Series B*, 64:583–639, 2002.

[3] JR Gonzalez, C Abellan, and JJ Abellan. A bayesian shared component model to analyze copy number data in genetic studies. *Statistics in Medicine*, submitted, 2010.

[4] M. Plummer. Penalized loss functions for bayesian model comparison. *Biostatistics*, 9(3):523–539, 2008.

[5] Martin Plummer. JAGS version 1.0.3 manual. Available at http://calvin.iarc.fr/ martyn/software/jags/, April 2009.