1 Tutorial on Using R Code

This tutorial explains how to use the R code presented in the paper:

Skelly DA, Johansson M, Madeoy J, Wakefield J, Akey JM (2011) A powerful and flexible statistical framework for testing hypotheses of allele-specific gene expression from RNA-Seq data. *Genome Res* 21:1728-37

1.1 Preliminaries

Before you can use the R scripts described below as intended, you must install the optparse R package found at http://cran.r-project.org/.

1.2 R Scripts

For the analysis you will need two main R scripts and one "accessory" R script.

1. DNAmodel.R - implements the model for read counts derived from genomic DNA, which is used to estimate parameters for the RNA read count model.

2. RNAmodel.R - implements the model for read counts derived from cDNA, which is used to identify allele-specific expression.

3. readGzippedMcmcOutput.R - script that can be sourced from within an R session and is used to read the saved output of scripts 1 and 2.

1.3 Input format

The read counts that must be fed into DNAmodel.R and RNAmodel.R should consist of read counts at each SNP, not aggregated by gene. The file must be in the following tab-delimited format:

# comments (ignored)			
# next line is header which is also ignored			
geneNameColumn	snpIndexColumn	alleleOneCount	alleleTwoCount
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮

where leading lines beginning with a # are ignored as comments, `geneNameColumn` names the first column of genes, `snpIndexColumn` is a column used to give a unique name/numerical value to each SNP (not currently used in the scripts), and `alleleOneCount` and `alleleTwoCount` are just the read counts themselves.

1.4 Basic Usage

The scripts are designed to be used from the command-line in a unix-like environment, but could easily be adapted for Windows usage. `DNAmodel.R` and `RNAmodel.R` have command-line flags that give the basic details for usage. The arguments for `DNAmodel.R` are:

1. name of infile - genomic DNA read counts
2. name of outfile - the file in which to store results of MCMC. File will be gzipped.
3. number of iterations of MCMC to run
4. value of "`thin`" - one out of every `thin` iterations, the parameter values will be written to your outfile
5. number of scaling iterations - the proposal distributions can be scaled to achieve an acceptance rate of (by default) approximately 30%. The tuning will happen after blocks of MCMC of this number of iterations. This number does not count towards the number of MCMC iterations specified above, and the parameter values during the scaling iterations are not saved.

For details on the arguments to `RNAmodel.R`, run the program with no arguments (`./RNAmodel.R`). Your options are similar to those for `DNAmodel.R`,

but can be specified in a slightly more flexible manner. The general look of a command to run this program is
`./RNAmodel.R [options] dataset1 [dataset2 dataset3 ...] outfile`
where the number of input datasets can vary but must be at least one. As stated above, the script `readGzippedMcmcOutput.R` is designed to be `source`d from within an `R` session and is used to read the saved output of scripts 1 and 2.

1.5 Example Usage

Let's say we have collected one lane of genomic DNA read counts and four lanes of RNA-Seq data using the Illumina sequencing platform, and have calculated allele-specific read counts for each sample and collected them in the files `DNA.txt`, `RNA1.txt`, `RNA2.txt`, `RNA3.txt`, and `RNA4.txt`. A simple analysis of this data might be:

1. First, run an analysis of your genomic DNA data:
   `./DNAmodel.R DNA.txt DNAresults.gz 200000 40 2000`

2. Next, analyze the results of step 1 and obtain estimates of $\hat{a}$ and $\hat{d}$. *Note the warning below about checking for convergence.* Inside an `R` session, you could execute the commands:
   ```
   > source('readGzippedMcmcOutput.R')
   > result <- read.mcmc('DNAresults.gz')
   > n.iter <- result$n.iter/result$thin
   > burnin <- 0.1*n.iter
   > a.hat <- median(exp(result$mcmc$logA[burnin:n.iter]))
   > d.hat <- median(exp(result$mcmc$logD[burnin:n.iter]))
   ```

3. Run an analysis on your RNA data:
   `./RNAmodel.R --n.iter=500000 --thin=100 --n.scaling.iter=2000 --a.hat=2000 --d.hat=500 --max.rounds.of.scaling=8 RNA1.txt RNA2.txt RNA3.txt RNA4.txt RNAresults.gz`
   Note the warning below about checking for convergence.

4. Examine the results of the RNA run in `R`:
   ```
   > source('readGzippedMcmcOutput.R')
   > result <- read.mcmc('RNAresults.gz')
   ```

1.6 Extracting results

After you run the scripts as described above in sections 1.4-1.5, you will end up with a gzipped file of MCMC results (in section 1.5, this is named `RNAresults.gz`). From this file, the most useful quantities that can be obtained are the variables π_0; p_i and e_i for each gene i; and the probability that gene i shows allele-specific expression. For this latter quantity we will need to use the formula at the bottom of page 9 in the supplementary material, which requires values of p, e, f, g, h, π_0, $\hat{a}$, and $\hat{d}$.

To obtain estimates of these quantities we can execute the following commands inside of an `R` session:

```
> source('readGzippedMcmcOutput.R')
# define the inverse logit function
> inv.logit <- function(val) exp(val)/(1 + exp(val))
> result <- read.mcmc('DNAresults.gz')
> genes <- result$features     # character vector of gene names
> a.hat <- result$a.hat
> d.hat <- result$d.hat
# below we take the posterior median of MCMC samples that are vectors
of length equal to n.iter
> pi0 <- median(inv.logit(result$mcmc$logitPi0))
> f <- median(exp(result$mcmc$logF))
> g <- median(exp(result$mcmc$logG))
> h <- median(exp(result$mcmc$logH))
# below we take the posterior median across columns of matrices with
nrow==n.iter and ncol==length(genes)
> p <- apply(inv.logit(result$logit_p), 2, median)
> e <- apply(inv.logit(result$logit_e), 2, median)
```

Now we have all the quantities we need. `p` and `e` are vectors of length equal to the number of genes, so to get estimates of (say) p_{17} and e_{17} we just look at `p[17]` and `e[17]` in `R`. To get the posterior probability of ASE for a particular gene we can use the following formula (reproduced from page 9 in the supplement)

$$p = \frac{\texttt{dbeta}(p|f,g)\texttt{dbeta}(e|1,h)(1-\pi_0)}{\texttt{dbeta}(p|f,g)\texttt{dbeta}(e|1,h)(1-\pi_0) + \texttt{dbeta}(p|\hat{a})\texttt{dbeta}(e|1,\hat{d})\pi_0}$$

where `dbeta` is the beta density function.

We might also be interested in calculating the false discovery rate when calling a set of features significant. Say we have used the above formula to calculate the posterior probability of ASE for every gene (and we have stored this result in a numeric vector called `posteriorProbAse` (of length equal to the number of genes for which we measured expression levels). We can then calculate the overall false discovery rate when calling features with (say) posterior probability of ASE > 0.8 as significant with the following `R` code:

```
> cutoff <- 0.8     # change cutoff at will
> calledSignificant <- which(posteriorProbAse >= cutoff)
> fdr <- mean((1 - posteriorProbAse)[calledSignificant])
```

1.7 Sample data

I have compiled a small sample RNA dataset that you may wish to use to ensure that the `R` scripts are running properly. This dataset consists entirely of simulated data so the true values of all parameters are known. The dataset is 50 genes, where each gene has between 1 and 5 SNPs and the coverage at each SNP is Poisson distributed with mean 300. In this simulated data, half of the genes show ASE, and half do not show ASE. The following are the true values of each parameter:

$$\begin{aligned}\hat{a} &= 500\\ \hat{d} &= 1000\\ f &= 5\\ g &= 5\\ h &= 20\\ \pi_0 &= 0.5\end{aligned}$$

Note: these values have been chosen rather arbitrarily, so it is not necessarily bad if you see values that are quite different in your actual (non-simulated) data. I ran `RNAmodel.R` using this simulated data with the command:

```
./RNAmodel.R --n.iter=100000 --thin=100 --n.scaling.iter=2000
--max.rounds.of.scaling=5 --a.hat=500 --d.hat=1000 simulatedData50genes.txt
simulatedData50genes-out.gz 2> simulatedData50genes1.log
```

This took about 25 minutes to run on a 3.33 GHz CPU with 4 Gb RAM.

Executing the following code in an R session shows that the results are as we expect, i.e. the posterior distributions of parameters include their true values and inferred values of p_i and e_i for each gene are close to their true values.

```
> source('readGzippedMcmcOutput.R')
> inv.logit <- function(val) exp(val)/(1 + exp(val))
> result <- read.mcmc('simulatedData50genes-out.gz')
> hist(exp(result$mcmc$logF))
> hist(exp(result$mcmc$logG))
> hist(exp(result$mcmc$logH))
> hist(inv.logit(result$mcmc$logitPi0))
> load('trueValues.Rdata')
> p.true
[1] 0.6626116 0.5060690 0.4690048 0.6610554 0.3463875 0.5158258
[7] 0.5174206 0.5130136 0.4984910 0.5023504 0.5410948 0.5175285
[13] 0.5137415 0.5620388 0.5132943 0.1974173 0.5578723 0.6564482
[19] 0.3839517 0.4930944 0.7539602 0.4922985 0.6184599 0.4791786
[25] 0.5062900 0.4797017 0.5343286 0.5175784 0.4979721 0.3066247
[31] 0.7620101 0.5146986 0.4837929 0.5209062 0.4610464 0.7443861
[37] 0.5376732 0.3976954 0.5138220 0.4767660 0.6537073 0.5078267
[43] 0.2693251 0.4760823 0.6723698 0.6128043 0.7393318 0.5129163
[49] 0.5222207 0.7043429
> e.true
[1] 2.280807e-02 1.613213e-03 5.445363e-04 5.830355e-04 9.550718e-03
[6] 1.218790e-02 5.801549e-04 3.012189e-04 7.039235e-04 7.168069e-04
[11] 6.174186e-02 1.266529e-03 2.819844e-03 3.346149e-02 2.820236e-04
[16] 8.239054e-02 2.550545e-03 5.751159e-02 3.241195e-02 4.889295e-04
[21] 1.601494e-01 2.486705e-04 3.670602e-02 2.741677e-03 1.286787e-03
[26] 1.949170e-03 1.174512e-01 2.276314e-04 1.597160e-03 3.791292e-02
[31] 1.011558e-01 1.179732e-03 2.870782e-03 4.076936e-04 1.253305e-03
[36] 1.305254e-01 2.195357e-04 6.520508e-02 3.930946e-04 1.557770e-04
[41] 2.102296e-01 7.946806e-05 5.680259e-02 4.817029e-04 4.261961e-02
[46] 3.001975e-02 1.378645e-01 3.362148e-04 5.414582e-04 1.167085e-02
> p.mcmc <- inv.logit(apply(result$logit_p, 2, median))
> e.mcmc <- inv.logit(apply(result$logit_e, 2, median))
> cor(p.true, p.mcmc)
[1] 0.9562227
> cor(e.true, e.mcmc)
```

```
[1] 0.7886158
```

I ran the MCMC for much longer using the following command:

```
./RNAmodel.R --n.iter=1000000 --thin=1000 --n.scaling.iter=5000
--max.rounds.of.scaling=5 --a.hat=500 --d.hat=1000 simulatedData50genes.txt
simulatedData50genes-out.gz 2> simulatedData50genes1.log
```

This took slightly under four hours to run on a 3.33 GHz CPU with 4 Gb RAM. The results were in very close agreement with the results for the shorter run above. A discussion of the use of statistical methods for diagnosing convergence of MCMC is beyond the scope of this document, but the references below may be helpful for this task.

1.8 Warning!

MCMC can be dangerous! Make sure you understand something about MCMC before attempting to run these scripts. You should always run multiple chains from different starting parameters, and verify convergence by examining time series plots or by other more formal measures such as the convergence diagnostics proposed by Gelman and Rubin [1], Geweke [2], or Heidelberger and Welch [3]. These scripts do nothing to check that you have completed any of these tasks.

References

[1] Gelman A, Rubin D: **Inference from Iterative Simulation Using Multiple Sequences**. *Statistical Science* 1992, 7:457–472.

[2] Geweke J: **Evaluating the accuracy of sampling-based approaches to calculating posterior moments**. In *Bayesian Statistics 4*, edited by Bernardo JM, Berger JO, Dawid AP, Smith AFM. Clarendon Press, 1992, 169–193.

[3] Heidelberger P, Welch P: **Simulation Run Length Control in the Presence of an Initial Transient**. *Operations Research* 1983, 31:1109–1144.