User's guide to taxDecomp

Cavan Reilly*

October 31, 2014

1 Introduction

This user's guide provides documentation for using the program described in Pragman et al. (2014) for testing for differences in the quantity of the identified taxa in a metagenomics experiment. Throughout this document we use `$` for the linux prompt and `>` for the prompt in the R console. The software runs on any linux system: one can simply download the executable (called `taxDecompV1.0`), make it an executable on your system:

```
cavan@cavan-DX441S:~$ chmod +x taxDecompV1.0
```

and run the program from the linux prompt (details of usage are provided in Section 1.1). No additional libraries are necessary to run the program and provided the executable is installed in the directory from which one issues commands to run the program there is no need set any environmental variables. On the other hand, strict adherence to file formatting is necessary and all arguments must be specified at the command line and in the proper order (or you will encounter a segmentation fault). Some tips for formatting the necessary files in R are provided in Section 2. One can also download the source code (i.e. download `taxDecompV1.0.cpp`) and compile that using the following command.

```
cavan@cavan-DX441S:~$ g++ taxDecompV1.0.cpp -o taxDecompV1.0
```

The program uses Markov chain Monte Carlo (MCMC) to simulate draws from the posterior distribution of interest, hence one needs to specify a number of parameters relating to the details of how the algorithm proceeds. The 4 basic parameters that control how the program runs are:

1. the total number of iterations to use
2. the number of "burn-in" samples (a reasonable choice for this is half the total number of iterations)
3. the amount of "thinning" that one would like to apply to the generated samples.

*Division of Biostatistics, University of Minnesota, A448 Mayo Bldg., MMC 303, 420 Delaware St. SE, Minneapolis, MN 55455-0378. e-mail: cavanr@biostat.umn.edu

4. the covariance matrix of the Metropolis jump distribution.

Markov chain theory indicates that if the total number of iterations is arbitrarily large then the algorithm will eventually generate samples from the posterior distribution-this suggests that we should choose the total number of iterations to be a large number (like 10,000). Since the initial samples that the algorithm generates will be influenced by the starting values and will not be approximate draws from the posterior distribution, it is customary to simply discard the initial samples (as stated above, the first half of the samples is a convenient choice). For some problems one could need a very large number of iterations for the algorithm to generate samples from the posterior-this typically happens when the data from many microbes includes many zeros. While some filtering of metagenomic data sets makes sense there can still be many zeros and this can make the algorithm take a long time to converge. In this case we may need say a million samples, and storing that many samples on one's computer can take a lot of space, hence it makes sense in these cases to only save every 10^{th} or 100^{th} iteration. The process of only saving a fraction of the samples is called thinning and one must supply a value for this at the command line: a value of 1 corresponds to no thinning, while positive integers greater than this indicate that only a fraction of samples are saved (a value of 10 would save every 10^{th} sample). You should ensure that the number of iterations less the number of burn-in samples is divisible by the thinning factor (e.g. 10,000 samples with 5,000 burn-in samples and thinning by a factor of 10 is fine but thinning by a factor of 9 would throw an error). Another option for use of the program is to specify if one wants to save all of the samples generated by the Markov chain (after thinning) or simply the posterior probability that the mean is lower in group 1 than group 2 for each identified taxon. This output can be interesting to explore and also can help with ensuring the program is working well (see the section below on the covariance file, Section 2.3). These results will be saved to a file called `postSmpl.txt`, and if you run the program again this file will be written over. Note that the parameter values will be on the log scale in this file, and that all parameters are sampled on the log scale.

In addition to specifying the details of how the MCMC algorithm works, three input files are also necessary. One should provide information on group membership for each subject and the number of reads that map to each genus for each sample (we call this the *data file*). The second file should have information on the taxonomic classifications for each genus (we call this the *taxonomy file*). The final file should have the entries of the covariance matrix for the Metropolis algorithm (we call this the *covariance file*). All of these should be tab delimited plain text files-details on each of these are provided in Section 2.

1.1 Usage

Here is an example of how to run the program: we select to save all of the output (the first 1 after the 3 input files), use 100 total iterations, discard the first 50 and save every sample after the burn-in period (i.e. perform no thinning).

```
cavan@cavan-DX441S:~$ taxDecompV1.0 copdBactCntsF1.txt copdBactTaxaF1.txt cov.txt 1 100 50 1
```

If we wanted to collect more samples, say a million, and save only every 1000 after discarding the first half we would use the following command.

```
cavan@cavan-DX441S:~$ taxDecompV1.0 copdBactCntsF1.txt copdBactTaxaF1.txt cov.txt 1 1000000 500000 1000
```

If we wanted to run this but not save the samples (which would save some space on your computer) you would issue the following command.

```
cavan@cavan-DX441S:~$ taxDecompV1.0 copdBactCntsF1.txt copdBactTaxaF1.txt cov.txt 0 1000000 500000 1000
```

Most applications will entail something between these 2 extremes. For example, the following settings were used for the simulation studies presented in the paper and will likely work for most applications.

```
cavan@cavan-DX441S:~$ taxDecompV1.0 copdBactCntsF1.txt copdBactTaxaF1.txt cov.txt 1 10000 5000 1
```

2 Details on input files

Most errors will likely be attributable to improper formatting of input files, so be sure that your files are formatted properly and listed in the correct order on the command line. Here are details on this.

2.1 The data file

The data file should have a row for each subject. Each row should have the same number of fields with the first field indicating which group each subject belongs to-these should be either 1 or 2. The remaining fields indicate how many reads from each sample map to each genus identified by your read mapping strategy. Here we suppose that we have a CSV file that holds a row for each genus and a column for each sample with the first row indicating group membership and the first column indicating genus (I frequently get excel files in this format, so just save the file as a CSV file in excel). Here is some R code for reading this in and setting up the data file.

```
> copdCnt <- read.csv("copdBactCnts.csv",header=FALSE)
```

This example has phenotypes in first row, some say "Control" followed by other information, so just label the cases 1, for example

```
> phen <- as.numeric(substr(as.matrix(copdCnt)[1,],1,3)!="Con")
```

Then write out counts with phenotype indicators (they need to be 1 and 2 so we add 1 to those) in the first column and strip off the taxa names.

```
> write.table(t(rbind(phen[-1]+1,as.matrix(copdCnt)[-1,-1])),"copdBactCntsF1.txt",
+ quote=FALSE,sep="\t",row.names=FALSE,col.names=FALSE)
```

Note that we transposed the contents of the original file as we want a row for each subject and that file had a column for each subject. For this example we are not quite done since we need to format the taxonomy file and this entails changes to the data file-see the next section.

2.2 The taxonomy file

We were able to obtain an excel file with the taxonomic information for all genera from the RDP website. We start by saving this as a CSV file then read that into R.

```
> copdTaxa <- read.csv("copdBactTaxa.csv",header=FALSE)
```

First we check that the taxa are in same order as in the data file-the first column of each holds genus names, so we can check for consistency with the following:

```
> sum(copdTaxa[,1]!=copdCnt[,1])
```

which should give 0. We then need to clean up the contents of this file-some taxa names are surrounded by quotes while others aren't: this will fix that up.

```
> taxMat1 <- gsub("\"","",as.matrix(copdTaxa)[-1,])
```

For our data set some of the phyla/genera combinations had no intermediate taxa due to these being newer taxonomies. We just get rid of these as it is not clear how to define intermediate taxa for these cases. So we check if any family names are blanks and get rid of those rows as follows.

```
> which(taxMat1[,2]=="")
[1] 51 110
> taxMat1 <- taxMat1[-c(51,110),]
```

Our file had white space before and/or after the names for some taxonomies-here we just get rid of this whitespace. Note that some taxa have spaces internal to their names so this will concatenate all portions of those names.

```
> taxMat2 <- gsub(" ","",taxMat1)
```

Then write the result to a tab delimited file:

```
> write.table(taxMat2,"copdBactTaxaF1.txt",sep="\t",quote=FALSE,row.name=FALSE,
+ col.names=FALSE)
```

But wait-this changed the number of genera under consideration, so we need to delete these from our data file and rewrite to file.

```
> write.table(t(rbind(phen[-1]+1,as.matrix(copdCnt)[-c(1,51,110),-1])),
```

```
+ "copdBactCntsF1.txt",quote=FALSE,sep="\t",row.names=FALSE,col.names=FALSE)
```

2.3 The covariance file

The goal of setting up a covariance matrix for the Metropolis algorithm is to get a Markov chain that makes large moves in parameter space and accepts about 30-40% of the moves. The proportion of moves accepted by the algorithm is printed to the screen at the end of the simulation runs, so one can examine how the proportion of accepted moves depends on the covariance matrix. There needs to be a dimension for each model parameter in this covariance matrix-the number of parameters in the model is 3 times the number of genera plus 2-so here we suppose that the R variable `ngen` holds the number of genera. A simple selection is to just try a diagonal covariance matrix with 0.1 on the diagonal-here is how to set that up in R:

```
> ngen <- dim(copdCnt[-c(1,51,110),-1])[1]
> write.table(diag(rep(0.1,3*ngen+2)),"cov.txt",row.names=FALSE,col.names=FALSE)
```

If you run 100-1000 iterations (which should only take a second) you will see the proportion of accepted moves printed to the screen after the algorithm finishes. If this is too large (i.e. bigger than 0.4) you should try a diagonal matrix with bigger values (say 0.5) and repeat. If the acceptance rate is too low (i.e. lower than 20%) then try a smaller diagonal matrix (say 0.01) and rerun.

Once you have run the algorithm you can use the covariance matrix of your samples to fine tune the performance of the algorithm if you wish. To do this, run the algorithm and save the posterior samples (set the first option to 1 in the command line), then read these into R and write the covariance matrix of these samples to file.

```
> res1 <- read.table("postSmpl.txt")
> write.table(var(res1),"cov.txt",row.names=FALSE,col.names=FALSE)
```

It can transpire that the resulting estimated covariance matrix isn't allowed (i.e. it is not positive definite) in which case the program will fail and issue an error message about the Cholesky decomposition not working (i.e. `choldc failed`). In this case you can add a diagonal matrix to this covariance matrix so that the Cholesky decomposition works for the result. Here we just add 0.01 to all diagonal elements.

```
> write.table((var(res1)+diag(rep(.01,3*ngen+2)),"cov.txt",row.names=FALSE,
+ col.names=FALSE)
```

If the acceptance rate is too low or too high you can rescale this covariance matrix and try again, using R code like the following:

```
> write.table(0.1*(var(res1)+diag(rep(.01,3*ngen+2))),"cov.txt",
+ row.names=FALSE,col.names=FALSE)
```

An important thing to understand is that no covariance matrix is "wrong" however some will not give useful results for a given number of simulations. Moreover there are no incorrect choices for the other parameters that govern how the MCMC works, however for a finite number of samples some values give a poor approximation to the posterior. Future versions may include greater assistance with determining the quality of one's Monte Carlo approximation. Let me know if you think we should build that functionality.