Final Log – February 2015

Logan Brown

February 9, 2015

Contents

1 Status of the NSE Code

The NSE code is running fairly well, as far we can tell. We've run it with good success on four different genomes. We've run it with and without phi, and as far as we can tell, it seems to gather good information about codon usage bias.

We had to make a few changes to the code to capture the model.

1.1 stable_exp.c

stable_exp.c is a section of C code that does one of the more intensive portions of the code's runtime. By our latest profiling, it takes about 70% of the code's runtime.

my.lodgmultnomCodOne and stable_exp.c do the following calculation. For the NSE model, it's

$$Pr(Codon|Parameters) = \frac{\exp\left(\text{MutationBias} - \text{NSEprob} * \text{position} * \text{expression} * \text{cost}\right)}{\sum_{synonyms} \exp\left(\text{MutationBias} - \text{NSEprob} * \text{position} * \text{expression} * \text{cost}\right)}$$

Or, using the symbols

$$Pr(c_{ij}|\theta) = \frac{\exp\left(\Delta\mu_{c_{ij}} - \Delta\omega_{c_{ij}} * j * \phi * a_2\right)}{\sum_{k=1}^{l} \exp\left(\Delta\mu_{c_{kj}} - \Delta\omega_{c_{kj}} * j * \phi * a_2\right)}$$

my.logdmulitnomCodOne does the following, disguised as a matrix multiplication xm times baamat

$$LP(c_{ij}) = \Delta\mu_{c_{ij}} - \Delta\omega_{c_{ij}} * j * \phi * a_2$$

And stable_exp.c does

$$\frac{\exp(LP(c_{ij}))}{\sum_{k=1}^{l} \exp(LP(c_{kj}))}$$

One problem this creates is that $exp(LP(c_{ij}))$ sometimes goes over DBL_MAX. Then the math returns NA, which causes a failure in the R code during the acceptance and rejection phase. To fix this, we scale down the terms. Wei Chen's code scaled it down by repeatedly dividing the values by 2 until they were sufficiently small. This could occasionally crash the code, for unclear reasons. It's possible that the sum was going back over DBL_MAX, or that the division by 2 never successfully scaled the term. We fixed this by always scaling down by the maximum value. This forced all values less than 0, which forced all of the exponentiated values to be between 0 and 1.

1.2 Values for a1 and a2?

The full version of the model is actually slightly more complicated. There's a middle term added based on $(a_1 - a_2)$

$$Pr(c_{ij}|\theta) = \frac{\exp\left(\Delta\mu_{c_{ij}} - \Delta\omega_{c_{ij}} * \phi * (a_1 - a_2) - \Delta\omega_{c_{ij}} * j * \phi * a_2\right)}{\sum_{k=1}^{l} \exp\left(\Delta\mu_{c_{kj}} - \Delta\omega_{c_{kj}} * \phi * (a_1 - a_2) - \Delta\omega_{c_{kj}} * j * \phi * a_2\right)}$$

I've generally set $(a_1 - a_2) = 0$. For some genomes, I do $a_2 = 4$ and for some, I do $a_2 = 1$. This is troublesome. To the credit of the model, it doesn't make a very large difference anyway. If $a_2 = 1$, it simply fits $\Delta\omega$ values at four times larger than $a_2 = 4$. The CUB plots can also correctly adapt for this change. The only real change that it makes is when you use a simulated genome and compare the 'true' values to the generated values.

1.3 Visualization changes

I made several changes to the visualization functionionality.

1. New function 'true' parameters vs estimated parameters, for use with simulated genomes where ω and μ are known.
2. Position - I made a change that shows the codon usage bias at various positions. By default, it shows for 100th, 200th, 400th, and 800th position. The line starts as a solid line and gets increasingly more dashed.
3. Weighted Centers - The CUB plotting function divides up the phi values into bins based on the quantiles of the phi values. The code previously placed a point in the direct middle of the bin. I added a flag weightedCenters that instead places the x value of the point at the mean of all the phis in that bin.
4. $\log_{10}$ bins - The quantiles that determine which phi values are grouped together are normally calculated off the log scale. I added a flag that allows you to calculate these bins on $\log_{10}(\phi)$ values. Uuu

2 Status of Newton

The CUBFits code runs on Newton. We never got to really explore the costs and benefits of running the code in this way. I'm somewhat skeptical, because most of the parallelization happens at the Amino Acid level. This means there would be no benefit to using any more than 19 processors, and really, we could do that on Gauley. Even though we get diminishing returns on the advantages of adding more processors, presumably, we want to grab 200 processors or so and go through the code quickly. However, we can't.

Here are detailed instructions to run CUBFits on Newton.

2.1 How to Install Packages

```
[Newton]$ export R_LIBS="$HOME/path_to_cubsrc/Dependencies"
[Newton]$ R
> install.packages('doSNOW')
> install.packages('coda')
> install.packages('seqinr')
> install.packages('EMCluster')
> install.packages('VGAM')
> install.packages('psych')
> install.packages('getopt')
> q("no")
```

2.2 How to Install CUBfits

```
export R_LIBS_USER="$HOME/path_to_cubsrc/Dependencies/"
R CMD build cubfits --no-build-vignettes --no-manual
R CMD INSTALL cubfits_0.1-2.tar.gz --library=$HOME/cubfits
```

2.3 How to run CUBfits on Newton

Run 'qsub run.sge' in the cubmisc/R folder. It goes as follows (replace USER with your Newton username)

```
[Newton:zeta00 R]$ cat run.sge
#$ -N Cubfits
#$ -q medium*
#$ -cwd
#$ -pe openmpi* 32
#$ -v R_LIBS_USER=/lustre/home/USER/cubsrc/Dependencies/:/lustre/home/USER/cubfits

echo $R_LIBS_USER
nohup Rscript run_loganYeast.r
```

3 Status of Genome Generation

Genome generation seems to run well.

I generated several yeast genomes, two based on the same values from Preston's simulated yeast, and two based on output values from a previous CUBFits run.

I'm slightly concerned because I tried to generate my yeast genomes with $a_2 = 4$, but the runs that I do tend to fit $a_2 = 1$. That's worth looking into.

4 NSE Model on various genomes

I've included data on each of these genomes in the data foler.

4.1 What we know about the 2013 REU's simulated yeast

This genome runs perfectly. It fits mu, omega, and phi almost perfectly. Run with $a_1 = a_2 = 4$

4.2 What we know about Preston's simulated yeast

This genome runs confusingly. It has three or four codons that are wildly inaccurate, but the rest of it fits perfectly. We think this is an error in the prior.

4.3 What we know about my simulated yeast

Run with $a_1 = a_2 = 1$. That's confusing, but it works. The code runs, with or without phi. Running with phi, as expected, improves the phi convergence.

4.4 What we know about Brewer's yeast

It seems to run fine. As a real genome, it's hard to tell what a_2 should be. The phi correlation isn't as good as we'd like, it hovers around $R^2 = 0.5$.

5 Running Scripts

I've included all of the running scripts in the data folder. You can also get them online from

`https://github.com/ozway/cubmisc`