Work Log for September

Logan Brown

October 1, 2014

4 Week of September 22nd-26th

4.1 Goals for the Week

1. Write out the math from 9/19 (from the green notebook)
2. Get observed yeast phi values from REU data
3. Run NSE model with the Browser line when *phi* is proposed to see what is causing NaN
4. Ideally, an NSE patch to fix it (try NaN to NA?)
5. Disect NSE data structure

4.2 Progress/Notes

4.2.1 Write out the math from 9/19 (from the green notebook)

see genomeProb.tex and genomeProb.pdf

4.2.2 Get observed yeast phi values from REU data

4.2.3 Run NSE model with the Browser line when *phi* is proposed to see what is causing NaN

7/22: The local library has been built. I removed the & to ensure that browser() will activate, and and wrote in two checks in my.logdmultinomCodOne.r

```
if(TRUE%in%is.nan(lp.vec)) {    browser();      }
```

but it's taken nearly 8 hours to run!!! The run started at: 2014-09-22 10:24:55
I left at 18:15:, so I don't know what happens.
The run finished at 18:28. But... browser didn't launch? I think I'll have to go into R manually, and use

```
source("run_nsef.r")
```

This means I'll have to parse the command line arguments manually.

CAUGHT

```
[1] "ALERT ALERT ALERT lpProp has NaN!"
Called from: my.drawPhiConditionalAll(phi.Curr, phi.Obs, y, n, b.Curr, p.Curr,
    reu13.df = reu13.df.obs)
Browse[1]> ls()
 [1] "b"         "lpCurr"    "lpProp"    "n"         "p.Curr"    "phi.Curr"
 [7] "phi.Obs"   "prop"      "reu13.df" "y"
There were 50 or more warnings (use warnings() to see the first 50)
Browse[1]> warnings()
Warning messages:
1: In tmp.phi * reu13.df.aa$Pos :
  longer object length is not a multiple of shorter object length
2: In tmp.phi * reu13.df.aa$Pos :
  longer object length is not a multiple of shorter object length
3: In tmp.phi * reu13.df.aa$Pos :
  longer object length is not a multiple of shorter object length
```

This means that it's actually not catching in my.logdmultinomCodOne.r

It catches in my.drawPhiConditionalAll. ~~but lpProp comes from my.logPosteriorAll.lognormal_bias. Which comes from~~ .cubfitsEnv does not correctly get MY functions. So it never got my.lodgmultinomCodOne with the browser() commands. Interesting.

I've done two separate runs of nsef cubfits.

1. one of the elements of lpProp is going to NaN.

2. It is not always the same element. For my first run, it was zraS. The second was ecnA.

3. It is not because log(0)= $-\infty$. Multiple elements are going to -Inf (106 in the first run, 87 in the second run). Moreover, the roc code also tends to generate -Inf values (though not nearly as many in each run, only 1 or 2).

4. It doesn't appear that the scale is going out of control. Scale and acceptance rate stay similar to the values used in the ROC model.

5. It happens in cubfits and cubappr

6. It is not just happening for one amino acid. For the first run, it happened in Amino Acid 5 (F). In the run, it happened in both 8 and 9 (I and K). The third run happened in 11 (N).

7. It is lp.vec, the return from my.inverse.mlogit.r

8. my.inverse.mlogit passes NON NaN values (though they are stupidly large like 1.452498e+18 instead of -0.5610390) to invmlogit, and it returns NaN values.

9. The code gets stuck in the following loop

   ```
   if(tmp_exp == HUGE_VAL || tmp_exp == 0.0){
   *flag_out_range = 1;
   *scale_exp = (tmp_exp == HUGE_VAL) ? max_exp : -max_exp;
   do{
   *scale_exp *= 0.5;
   tmp_exp = exp(*scale_exp);
   } while(tmp_exp == HUGE_VAL);
   *scale_exp = max_exp - *scale_exp;
   }
   ```

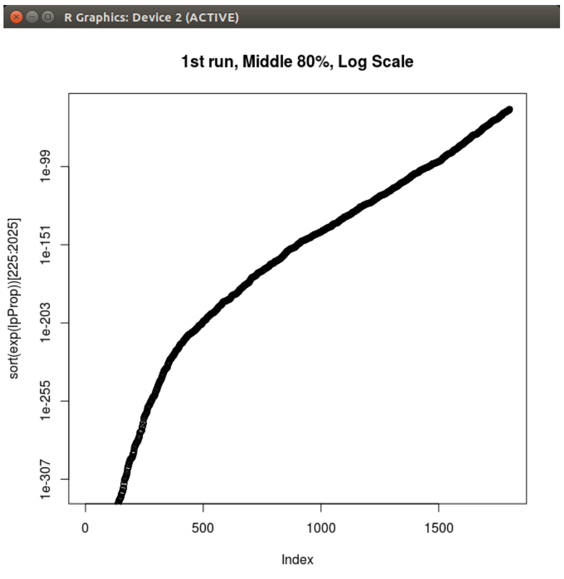
   That's what causes the slow down.

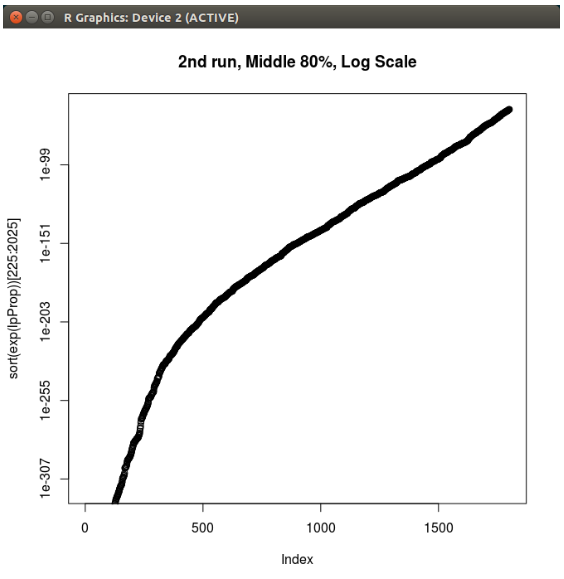
   But this means the problem comes earlier. At some point in the code, some probabilities are going to infinity, which causes the HUGE_VALUE loop (and the slowdown), and eventually causes NaN values.

10. All the values for lpProp are generally too low.
    mean(lpProp[is.finite(lpProp)]) returns -588.3597 in from the first run, and -554.7103 in the second run.

    Taking that out of the log scale (using mean(exp(lpProp[is.finite(lpProp)])) instead) gives 1.10973e-11 for the first and 1.085067e-12 for the second.

    Below is the information on a log scale. Note that while the probabilities were initially also on a log scale, they have been exponentiated. Also, note that the information doesn't actually follow any kind of a trend. I sorted the data in order to take off the top and bottom 10%.





Unfortunately that's all the information I was able to get because the terminal crashed.

Rerunning using GDB on R. When I hit the browser (when lpProp has NaN), I should be able to launch the C code using GDB and find EXACTLY the error.

4.2.4 Ideally, an NSE patch to fix it (try NaN to NA?)

4.2.5 Disect NSE Data Structure

Position information? is in reu13.df$(amino acid)$Pos

4.2.6 Rstudio / R vim extension?

4.2.7 Reread Debugging Chapter

Read the section on GDB. Trace seems flexible but not horribly useful in my case.

How to GDB an R session

1. cd /cubfits/misc/R
2. R -d gdb
   GDB will launch
3. run
   (R will run)
4. source("debug_nsef.r")
   NSE model will begin in the R session. Run until the R browser catches an error
5. Ctrl-C
   This returns to GDB (R is still active, but not running).Set a breakpoint.
6. continue
   R continues running with GDB breakpoint. Launch the code you want to analyze with GDB (and possibly browser)

4.3 Goals for next Week

1. Find out what causes some of the probabilties are going outrageously high
2. If possible, NSE Patch
3. More information on Codon Position
4. Look into an R Vim extension or RStudio
5. (Optional) Find out what trace() does