Work Log for September

Logan Brown

September 22, 2014

3 Week of September 15th-19th

3.1 Goals for the Week

1. Simulated Data Set
2. cubappr SimuYeast Run
3. First Order Approximation
4. Potentially look at the c files in cubfits/cubfits/src
5. Investigate WeiChen NSE crash
6. Compare NSE code to ROC code
7. Build Local Cubfits
8. Disect NSE data Structure
9. Profile the ROC model (where is the time?)
10. Profile the NSE model if possible
11. Continue to improve the optimal-pessimal code as Deepika works with it.

3.2 Progress/Notes

3.2.1 Simulated Data Set

The data is in /home/lbrown/reucode/data/inputsimdata/REU_data

As I understand it, these fasta genomes are ones that have been modified by Codon Evolution Simulation (CES). A quick comparison of b-1/S.cerevisiae.S288c.REU.sim.b-1.ces.fasta and ../S.cerevisiae.S288c.fasta shows that they have different codons. Additionally b-1's fasta file and b-0.01's fasta are different as well.

The folder b-1 vs b-0.1, etc is the setting of the B parameter. Look at '/export/home/clandere/CodonUsageBias/NSE/ces3/branches/exchange/C/data_simulation/CES.DATA.SIM for more details

I chose to use b-0.001, it had the best signs of actually converging to something. The eta values of the genes actually changed. For some b values, there wasn't an eta change. b-0.001/S.cerevisiae.S288c.REU.sim.b-0.001.evol.summary.tsv was the highest B value that had changes at every genome (Evolution Time != nan).

I was not able to find X_obs values for the yeast data in the REU data. I found ORF data in /opt/big_scratch/work-my, which is data from Wei-Chen/Yassour. I ran a recursive search through those files looking for xobs values, the output is found in smallfindXobs.txt

3.2.2 cubappr SimuYeast Run

Launched a cubappr run of the simulated yeast genome from the REU data, both for single chain and ~~multichain~~ (nothing happened for 2 hours. either it crashed, or changing the config.r file messed with the actual inner workings). If either/both crashes, I'll try again with a smaller run, likely just singlechain.

3.2.3 First Order Approximation

$$\prod_{j=i+1}^{n}(1-p_j) = \prod_{j=i+2}^{n} -p_{i+1}\left(\prod_{j=i+2}^{n}(1-p_{i+1})\right)$$

Simply to make things easier to read, I'll restate $\prod_{j=q}^{n}(1-p_j)$ as t_q. Note that in general,

$$t_q = t_{q+1} - p_q(t_{q+1})$$

So

$$\begin{aligned}\prod_{j=i+1}^{n}(1-p_j) &= t_{i+2} - p_{i+1}t_{i+2}\\ &= t_{i+3} - p_{i+2}t_{i+3} - p_{i+1}\left(t_{i+3} - p_{i+2}t_{i+3}\right)\\ &= t_{i+3} - p_{i+2}t_{i+3} - p_{i+1}t_{i+3} - p_{i+1}p_{i+2}t_{i+3}\end{aligned}$$

Since $p_{i+1} * p_{i+2} \approx 0$.

$$\approx t_{i+3} - p_{i+2}t_{i+3} - p_{i+1}t_{i+3}$$

Continuing iteratively...

$$\begin{aligned}\prod_{j=i+1}^{n}(1-p_j) &\approx t_n - \sum_{k=i+1}^{n-1} p_k(t_n)\\ &\approx (1-p_n) - \sum_{k=i+1}^{n-1} p_k(1-p_n)\\ &\approx 1 - \sum_{k=i+1}^{n} p_k\end{aligned}$$

3.2.4 Potentially look at the c files in cubfits/cubfits/src

They take up the majority of the time (see the profiling below)
lp_c_raw

3.2.5 Investigate WeiChen NSE crash

- Ending crash caused by running out of memory?

Cedric also suggests it could be due to a problem with the serialization step. When the code is run in parallel (by SNOW activating multiple copies of the same program with different initial conditions), it comes back to the serial to be tested for convergence. If the data structures are too large at this point, it may crash due to "running out of memory", even though Gauley has waaay more memory. It may be related to memory.c

I'm running in single chain from here on out for testing this hypothesis.

Single chain crashed, however, it actually told the error!

```
Error in phi.New[accept] <- prop$phi.Prop[accept] :
  NAs are not allowed in subscripted assignments
Calls: system.time ... do.call -> <Anonymous> -> my.drawPhiConditionalAllPred
Timing stopped at: 6251.473 43.7 6298.546
Execution halted
```

3.2.6 Compare NSE code to ROC code

Here are all the files that use the NSE model (results of grep -il nse /cubfits/cubfits/R/*)
~~my.coef.r~~

my.estimatePhiOne.r
my.fitMultinomOne.r

Here is where the vglm concerns are. Mostly not concerned, it doesn't look like any additional vglm calls.

my.logdmultinomCodOne.r

Here's my biggest concern.

my.logdmultinomCodOne.roc has three lines that my.logdmultinomCodOne.nsef does not lp.c.raw <- yaa * lp.vec lp.c.raw[is.nan(lp.c.raw)] <- NA lp.c.raw <- rowSums(yaa * lp.vec, na.rm = TRUE)

my.drawPhiConditionalAllPred calls my.logPosteriorAllPred.lognormal, which calls my.logdmultinomCodOne.nsef, which does not have the stated lines. Without those lines, if lpProp - lpCurr - prop$lir becomes NaN (log of a negative value, most likely), it would not pass any errors until the line that actually complains, accept <- u < exp(logAcceptProb).

However logAcceptProb is lpProp - lpCurr - prop$lir

My best hypothesis is that lpProp (proposed phi values from the Posterior distribution) has some NaN values.

CONFIRMED,

```
- var.name: p
    scale bound reached #: lower = 0, upper = 0
    ill acceptance #: none = 0, all = 0
    acceptance NOT in range #: lower = 0, upper = 0, total = 0
- var.name: phi.pred
    scale bound reached #: lower = 0, upper = 0
    ill acceptance #: none = 0, all = 0
    acceptance NOT in range #: lower = 664, upper = 720, total = 1384
[1] "ALERT ALERT ALERT lpProp has NaN!"
Error in phi.New[accept] <- prop$phi.Prop[accept] :
  NAs are not allowed in subscripted assignments
Calls: system.time ... do.call -> <Anonymous> -> my.drawPhiConditionalAllPred
In addition: There were 50 or more warnings (use warnings() to see the first 50)
Timing stopped at: 3434.911 12.498 3448.359
Execution halted
```

lpProp is coming up with NaN. I suspect that negative phi values are being proposed, and $ln(-x)$ is NaN, and e^{NaN} is NaN, etc, etc, etc, NAs are not allowed in subscripted assignments.

I'll add browser() lines to my.logdmultinomCodOne.r and see what happens. I also removed the & from the end of the line to be sure the browser command will work.

Probably don't need to be considered, but left here for completeness

my.objectivePhiOne.Lfp.r
my.objectivePhiOne.nlogL.r
my.objectivePhiOne.nlogphiL.r
my.objectivePhiOne.phiLfp.r

my.pPropTypeNoObs.lognormal_bias.r
plotbin.r
plotmodel.r
simu.orf.r

3.2.7 Build Local Cubfits

1. cd path/to/cubfits/..
2. R CMD build cubfits (This creates cubfits_ver.si-on.tar.gz)
3. R
4. install.package("_______.tar.gz", lib ="place to install to", repos = NULL, type="source")
5. library(cubfits, lib.loc="place you installed to")

3.2.8 Disect NSE data Structure

3.2.9 Profile the ROC model (where is the time?)

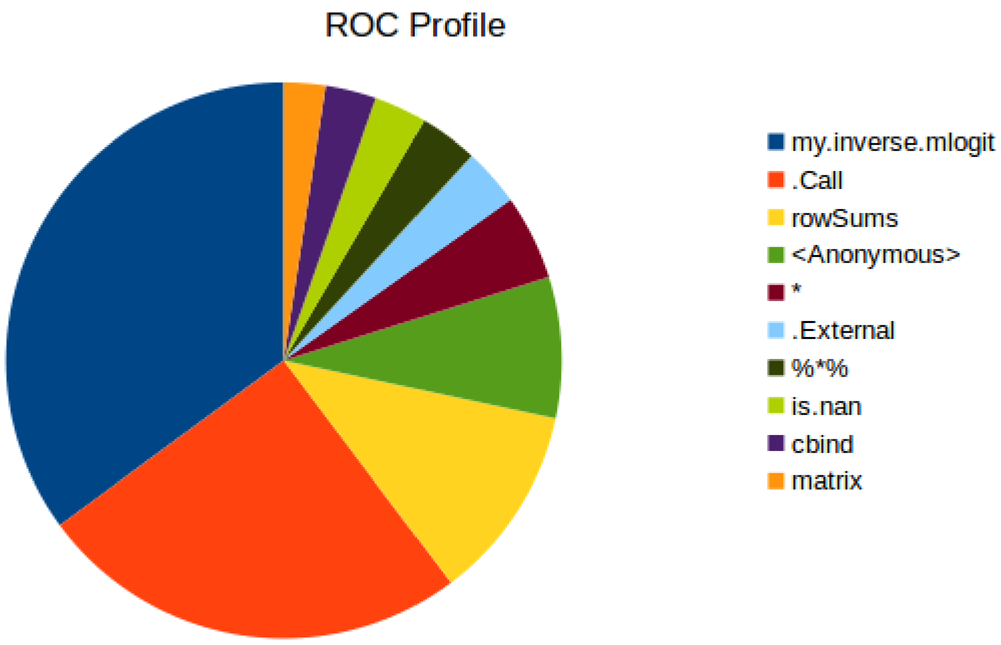
Look at the man page - ?Rprof

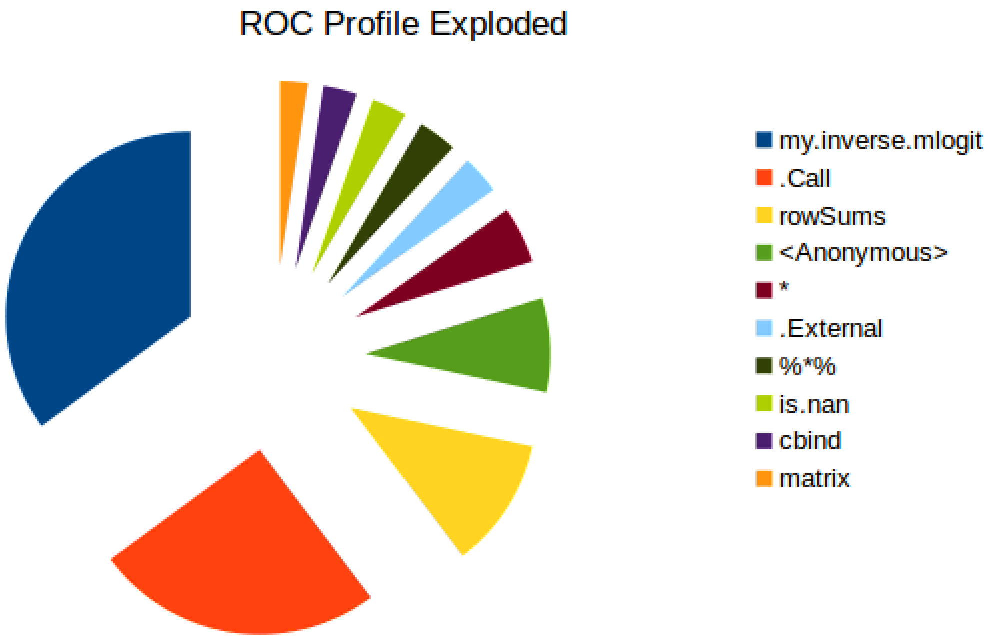
Run Rprof() before any code you want to profile, and Rprof(NULL) after the code, then run summaryRprof in R or R CMD Rprof from the command line to analyze the output.

Added Rprof() to the beginning of run_roc.r and Rprof(NULL) to the end of run_roc.r, then ran using ./workflow as usual.

These are the functions that took up more than 1% of the time on their own.

```
  %       self       %     total
 self    seconds   total  seconds    name
 33.3   2123.86    60.5   3855.96     "my.inverse.mlogit"
 23.9   1520.96    23.9   1520.98     ".Call"
 10.8    687.68    14.0    894.64     "rowSums"
  7.8    496.74    98.7   6288.94     "<Anonymous>"
  4.7    299.56     4.7    301.74     "*"
  3.2    205.44     3.2    205.44     ".External"
  3.2    203.82     3.2    203.82     "%*%"
  2.9    183.66     2.9    183.66     "is.nan"
  2.8    178.62     2.8    178.84     "cbind"
  2.3    143.80     4.6    294.34     "matrix"
```





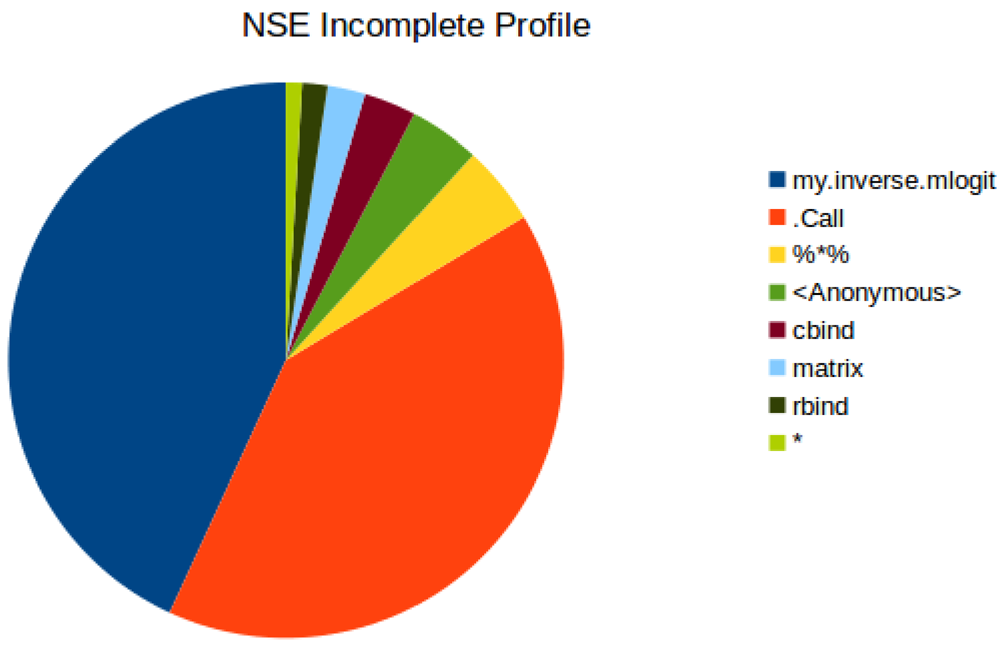
.Call is only seen in two places
.Call("invmlogit"... in my.inverse.mlogit.r
.Call("lp_c_raw"... in my.logdmultinomCodOne.r

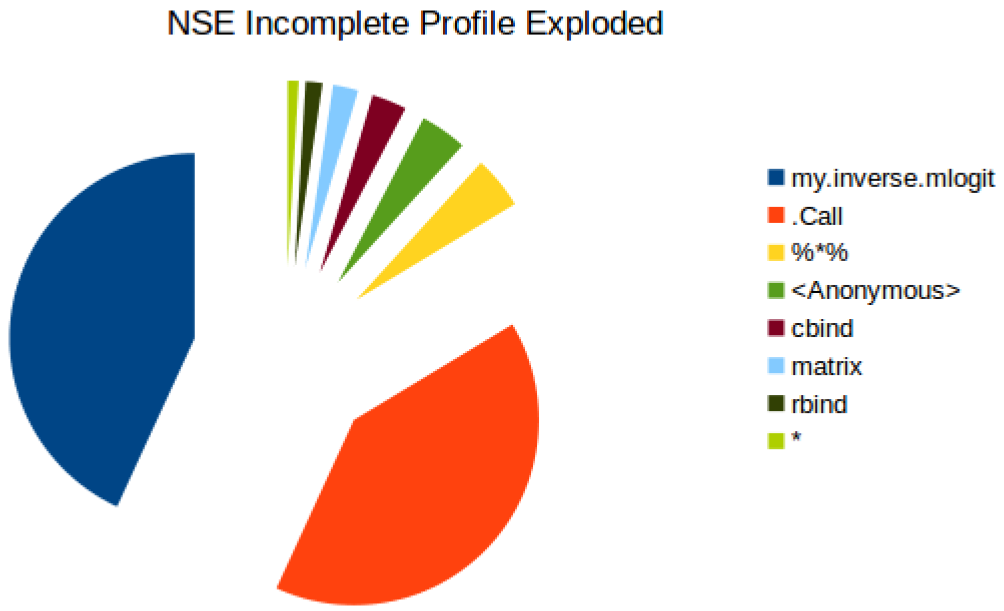
It's the function that calls the C code from the R code. The C code, apparently, is doing the bulk of the work.

3.2.10 Profile the NSE model if possible

I ran an incomplete run of the NSE model, crashing in the usual place, but profiled it.

```
  %        self       %      total
 self    seconds    total   seconds    name
 41.1   1851.12      80.7   3637.60     "my.inverse.mlogit"
 38.6   1741.60      38.7   1743.24     ".Call"
  4.4    199.78       4.4    199.78     "%*%"
  3.9    175.48      96.5   4347.20     "<Anonymous>"
  2.9    129.36       4.2    189.06     "cbind"
  2.1     96.60       6.3    282.04     "matrix"
  1.4     65.24       1.6     71.92     "rbind"
  0.9     41.30       1.3     58.36     "*"
```





3.2.11 Continue to improve the optimal-pessimal code as Deepika works with it.

3.3 Goals for next Week

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

2. Run NSE model with the Browser line when *phi* is proposed to see what is causing NaN
3. Ideally, an NSE patch to fix it (try NaN to NA?)