Work Log for September

Logan Brown

October 1, 2014

FOREWORD: I think for October, I'll just do the monthly summary. The things I'm doing are now carrying over well from week to week.

Contents

1 Week of September 1st-5th

1.1 Goals for the Week

1. Connect codons with Δeta values
2. Liz Howell Code
3. Further readings in the R user manual
4. NSE Model

1.2 Progress/Notes

1.2.1 Connect codons with delta eta values

I made a change to run_roc.r, line 314. I changed
mean.b.mat <- cbind(bmat.names, mean.b.mat, sd.b.mat)
to
mean.b.mat <- cbind(names(results$chains[[1]]$b.Mat[[1]]), mean.b.mat, sd.b.mat)

bmat.names is just the synonym group of the codon. names(results$chains[[1]]$b.Mat[[1]]) is of the format aminoacid.codon.value (for example A.GCC.log.mu or H.CAC.Delta.t where log.mu represents the natural log of the mutation rate and Delta.t represents the change in the pausing times (compared to some reference codon). This information gets written to

A note on the reference codon: Cedric claimed the one with the shortest pausing time was the reference codon, I've found it's the last one alphabetically.

1.2.2 Liz Howell Code

The code seems to be done. It can be checked out from github at

https://github.com/ozway/optimal-pessimal.git

It takes in a gene (by default, named "ecoli.fasta"), and can write the optimal or pessimal version(s) of the genome using makeOptimal.r, makePessimal.r, or makeBoth.r. The optimal is written to optimalEcoli.fasta, and the pessimal is written to pessimalEcoli.fasta. writeOptimal and writePessimal takes about 5 minutes and 49 seconds to run on the ecoli gene. writeBoth takes about 7 minutes and 45 seconds to run on the ecoli gene, likely due to the increased file writing.

Future Considerations:

1. ~~Add a config.r file for setting global variables like input filenames and output filenames)~~ Done
2. Calculate the total amount of time gained or lost in the changes
3. Do we need to switch from ROC to η?

4. Try for other genes?

I've also written up readme instructions for how to execute the code.

1.2.3 Further readings in the R user manual

In order to write the Liz Howell code, I had to do more readings into the R user manual. I read deeply into, and played around with

- Matrices
- Lists
- String Manipulation
- the [] and [[]] operators
- for and while loops

1.2.4 NSE Model

Following Wei-Chen's instructions, I attempted to run a cubfits NSE model.

1. Change the cubmultichain and cubsinglechain calls on lines 188, 195, 205, 212, changing model="roc" to model="nsef"
2. Before each of those lines, add .CF.CT$model <- "nsef";

And then running it as normal.

9/5: I started the model at at 9:02. After generating very little output, it ended at 12:38. It parsed all the inputs and such, but generated no results.

2 Week of September 8th-15th

2.1 Goals for the Week

1. NSE Model

2.2 Progress/Notes

2.2.1 NSE Model – run using workflow.sh

These are the changes I've made to cubfits/misc to try and run the NSE model. (run_nsef.r is just a copied version of run_roc.r with the following exceptions)

- In run_nsef.r, changed model="roc" to model="nsef" in cubsinglechain and cubmultichain for both "cubfits" and "cubappr"
- In run_nsef.r, added .CF.CT$model <- "nsef"; before each each call of cubsinglechain and cubmultichain

- In run_utility.r, changed get.logL <- function(ret, data, model="roc") to get.logL <- function(ret, data, model="nsef")
- Changed workflow.r from
  - Rscript run_roc.r -c $cubmethod -s "0.5 1 2 4" -f $genome -p $empphi -o $folder -n $foutname -i $pinit >> $logfile &

    to
  - Rscript run_nsef.r -c $cubmethod -s "0.5 1 2 4" -f $genome -p $empphi -o $folder -n $foutname -i $pinit >> $logfile &

I also changed to cubsinglechain instead of cubmultichain, to try and simplify matters, but then the MCMC started throwing "acceptance out of range" at every step of the iteration. Started at 10:44, ran until 11:44.

I'll retry with cubmultichain. I had run a multichain version on 9/5, and it just stalled. However, I just added run_utility.r change

Latest settings that worked!
n.samples = 10
use.n.samples = 10
n.chains = 4
n.cores = 4
min.samples=50
max.samples=100

```
================================================================================
================================ START HEADER ==================================
================================================================================
Function call:
Rscript run_roc.r -c cubfits -s 0.5 1 2 4 -f ../data/ecoli_K12_MG1655_genome_filtered.fas

MCMC parameters:
Number of samples between checks: 10
Min samples: 50
Max samples: 100
Reset QR until: 0 samples
Thining: store every 10 iteration as sample
Swap 0 % of b matrix
Swap b matrix if L1-norm is < 0

Simulation parameters
Number of Cores 4
Number of Chains: 4
Parallel mode within chain: lapply
Samples used: 10
```

```
First 0  AAs removed from sequence
Sequences with less than 0 AAs ignored
List of AAs taken into account:
A C D E F G H I K L M N P Q R S T V W Y Z

Convergence criteria
Convergence test: Gelman & Rubin
Convergence criterium: Gelman Score < 0.15
Use every 1 sample for convergence test

================================================================================
================================= END HEADER ===================================
================================================================================


started at: 2014-09-08 13:56:25
using cubfits
reading sequences from file ../data/ecoli_K12_MG1655_genome_filtered.fasta
reading gene expression measurements (Xobs) from file
 ../data/ecoli_X_obs.csv
and compare to ORF list from FASTA file
 ../data/ecoli_K12_MG1655_genome_filtered.fasta
generating list of codon position in ORFs for each AA...
generating list of number of AA occurences per ORF...
generating list of codon counts per ORF...
generate initial phi using SCUO with sd(ln(phi)) values
- 0.5
- 1
- 2
- 4
running cubfits using cubmultichain
 with seeds: 68072 35014 78804 49768
4 slaves are spawned successfully. 0 failed.
Gelman score after sample: 11 5.24608844491114  test was performed on 6 samples
Gelman score after sample: 21 4.43002963732354  test was performed on 10 samples
Gelman score after sample: 31 4.15615605401198  test was performed on 16 samples
Gelman score after sample: 41 4.00798122092697  test was performed on 20 samples
Gelman score after sample: 51 3.87651221427067  test was performed on 26 samples
Gelman score after sample: 61 3.76115242256423  test was performed on 30 samples
Gelman score after sample: 71 3.68585493775202  test was performed on 36 samples
Gelman score after sample: 81 3.61050525774646  test was performed on 40 samples
Gelman score after sample: 91 3.52149378911106  test was performed on 46 samples
Gelman score after sample: 101 3.43585418976874  test was performed on 50 samples
```

```
Elapsed time for 4 chains doing  iterations on 4 cores was 30.2 min
process results...
saving results...
finished at: 2014-09-08 14:28:03
```

I was able to repeat these results using 1000 samples, (minimum of 500 samples), though it stopped after 500 samples. The Gelman score was 1.10581040617552. For brevity, the 500 sample log is not included in this pdf, but it is in the github.

2.2.2 Larger NSE Model

Ran from 17:22 to 19:12, then ended with no error messages. "Timing stopped at"

Is it possible that the model is OVERconverging? I had it doing 500 samples between checks, and it should succeed at doing 500 samples...

2.2.3 NSE Model – debug

Ran the contents of cubfits/demo/nsef.train.r in an R interactive session, adding in the line debugonce(cubfits)

As far as I can tell, the cubfits() function itself doesn't actually use the model or .CF.CT$model, I'll need ot look at drawP and drawPhi.

According to Cedric, it's worthwhile to look at cubfits/cubfits/R/my.logdmultinomCodOne.r. That function calls cubfits/cubfits/src/stable_exp.c and cubfits/cubfits/src/lib.c.

2.2.4 Optimal/Pessimal Code

Cedric made the change that had been discussed, where we switch from Δt values to $\Delta\eta$

In making that change, I also made several more changes.

- fixed a bug where the default codon for Q was wrong
- Changed from analyzing Δt to $\Delta\eta$ values. Made a mock up etaValues.bmat for example by just inverting the signs of the values
- made the language more clear for "optimal/best" versus "minimum/maximum"
- added a count and ratio for how many codons are actually being up/downgraded

Deepika contacted me about an error in the code. The beginning of the for loop (line 78 in the original, now line 79)

for(index in 1:length(sequence[[gene]]/3)){

to

for(index in 1:(length(sequence[[gene]])/3)){

The first one was causing a fractional value of index, which was making it do weird things, but mainly, it was making the code replace random chunks of genome, rather than accurately dividing the codon into groups of 3.

2.2.5 First Order Approximations

For completeness, I also wrote down how we derived the fitness function in fitnessHistory.tex

p_{ic} is the probability of a nonsense error at position i using codon c
p_j is the probability of a nonsense error at position j

First order approximation about $p_{ic} = 0$ is

$$f(p_{ic} \approx f(0) + f'(0) * (p_{ic} - 0) = p_{ic} \left(\left[\sum_{k=1}^{i} a_1 + a_2(k-1) \right] \left[\prod_{j=i+1}^{n} (\frac{1}{1-p_j}) \right] \right)$$

First order approximation about $p_{i+1} \approx p_{i+2} \cdots \approx p_n \approx 0$ is

$$f(p_j) \approx \left[\sum_{k=1}^{i} a_1 + a_2(k-1) \right] \left[\frac{p_{ic}}{1-p_{ic}} \right] + ((i+1) - n) \left[\sum_{k=1}^{i} a_1 + a_2(k-1) \right] \left[\frac{p_{ic}}{1-p_{ic}} \right] (p_j)$$

$$f(p_j) \approx f(0) + (f(0))(p_j)((i+1) - n)$$

Details are in approx.tex

2.2.6 Simulated Data Set from REU13

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

I think elongation rate refers to the odds $\frac{p_i}{1-p_i}$.

2.3 Goals for next Week

1. Simulated Data Set
2. Potentially look at the c files in cubfits/cubfits/src
3. ~~Code the first order approximations? Looking at my draw*.r for the fitness function~~
4. Check for problems in Wei-Chen's NSE code, specifically
   (a) ~~Unnecessary VGLM calls leading to longer run times~~ As far as I can tell, my.fitMultinomOne.nsef and my.fitMultinomOne.roc are using the same amount of vglm calls.
   (b) Ending crash caused by running out of memory
5. Continue to improve the optimal-pessimal code as Deepika works with it.

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?)

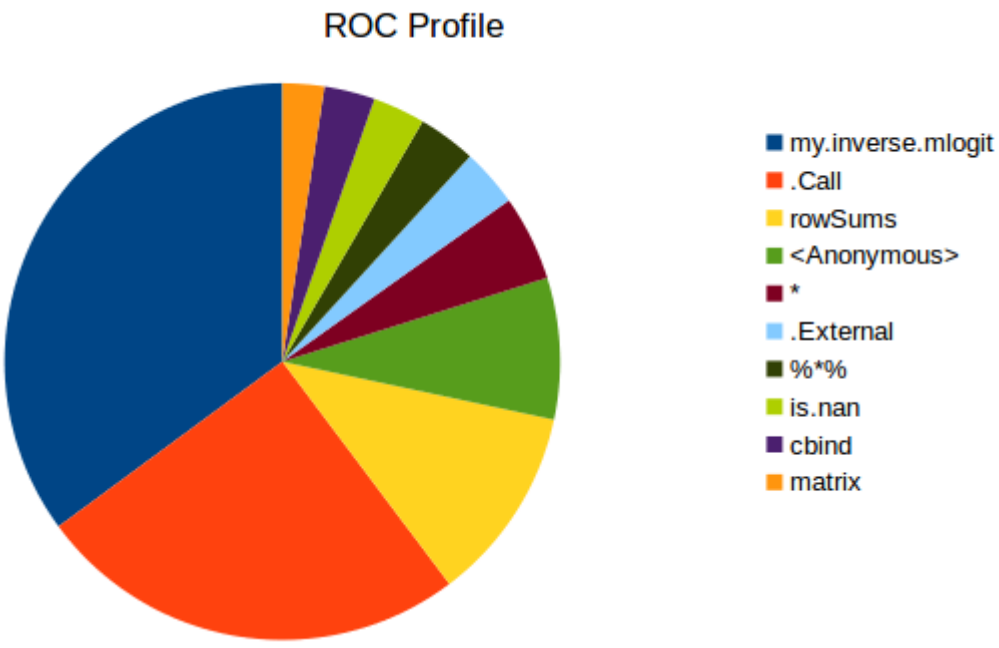
Look at the man page - ?Rprof

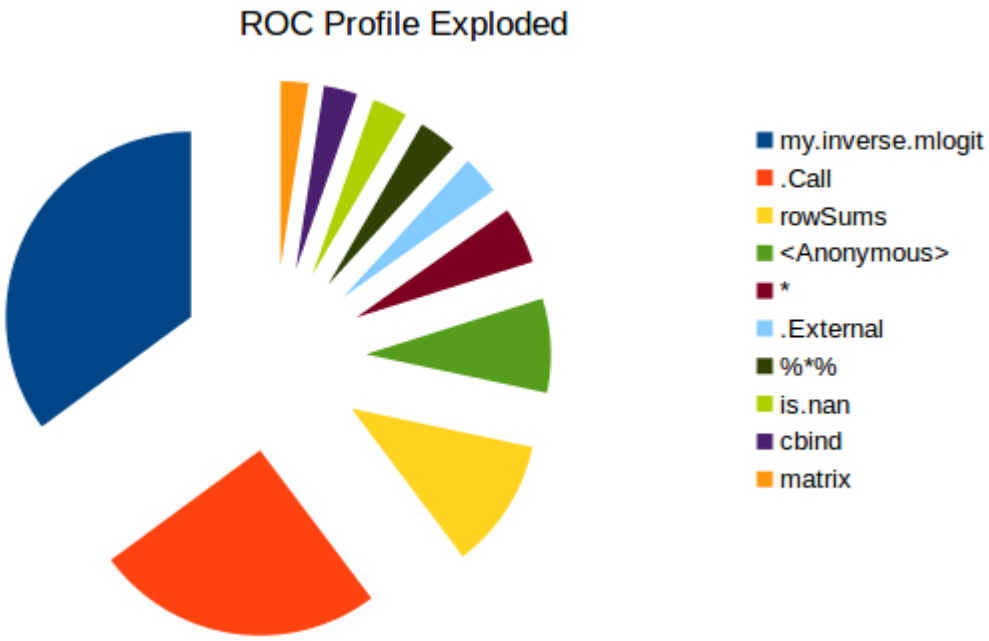
Run Rprof() before any code you want to profile, and Rprof(NULL) after the code, then run summaryRprof in R or R CMD Rprof from the command line to analyze the output.

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	self seconds	% total	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"





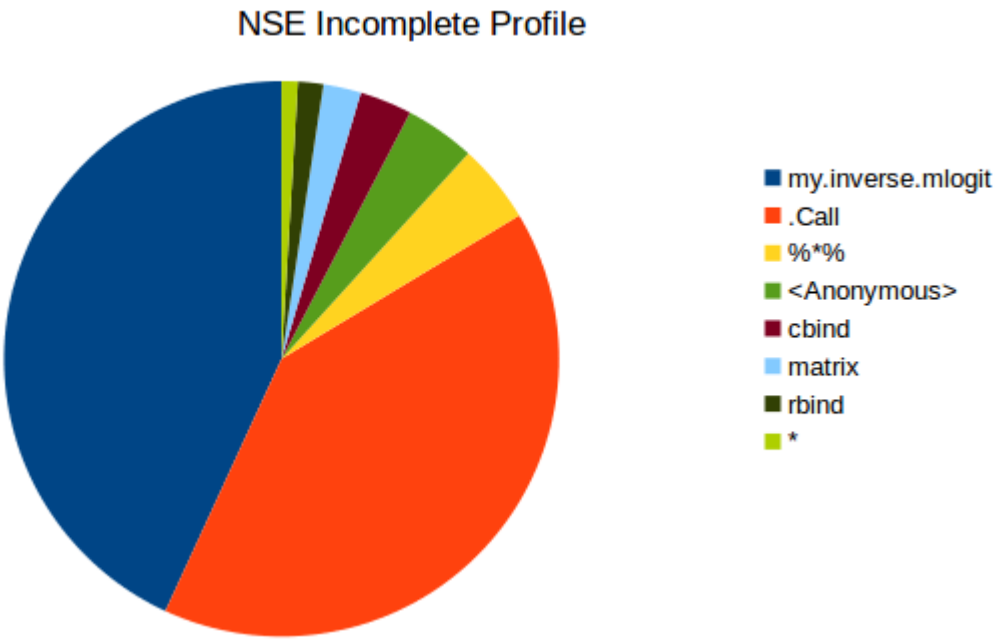
.Call is only seen in two places
.Call("invmlogit"... in my.inverse.mlogit.r
.Call("lp_c_raw"... in my.logdmultinomCodOne.r

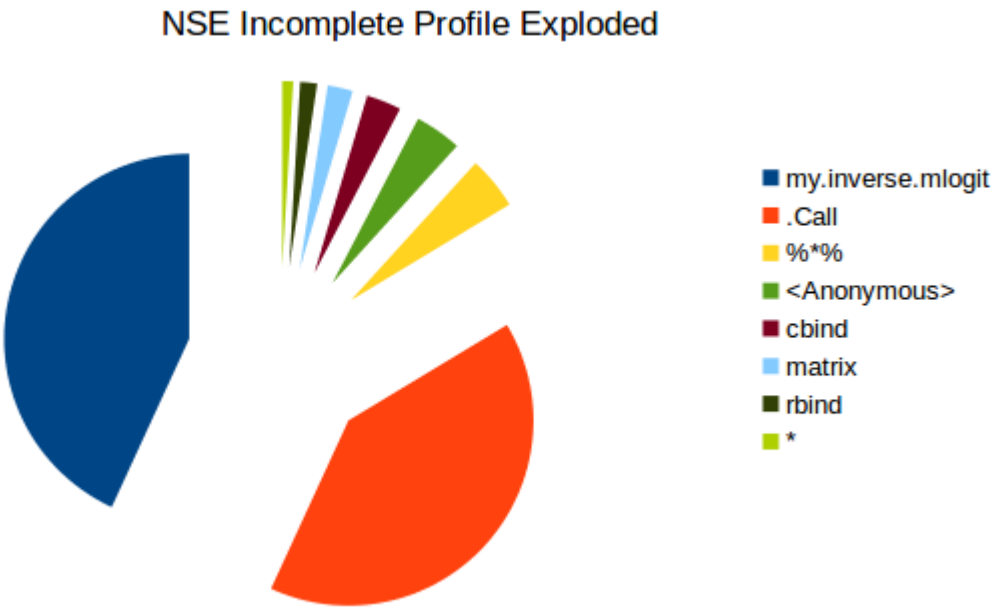
It's the function that calls the C code from the R code. The C code, apparently, is doing the bulk of the work.

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	self seconds	% total	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?)

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;
}
```

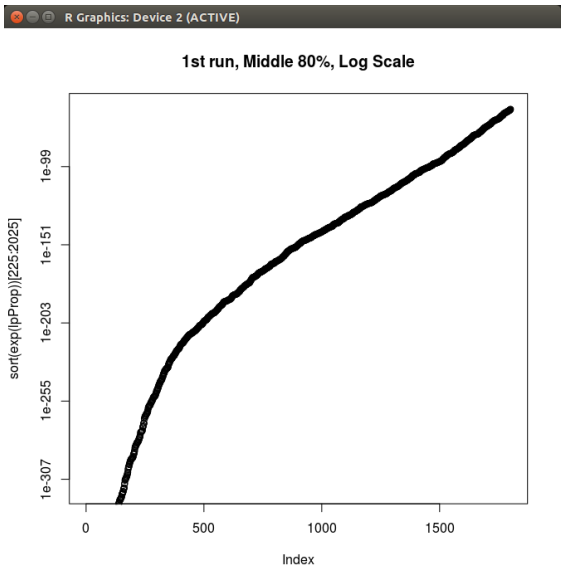
That's what causes the slow down.

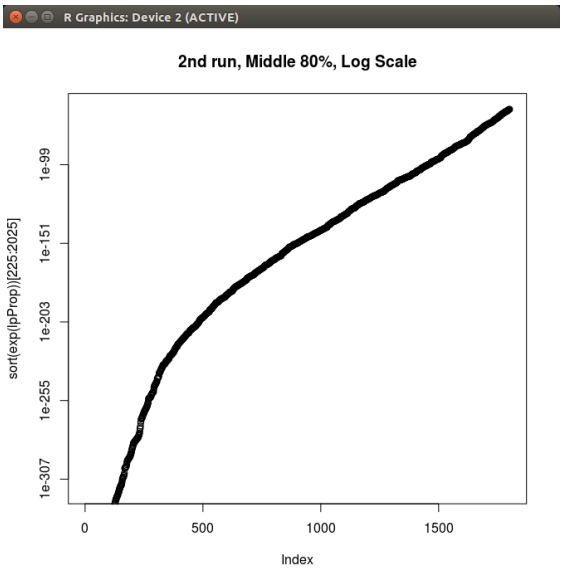
But this means the problem comes earlier. At some point in the code, some probabilities are going to infinity, which causes the HUGE_VALUE loop (and the slowdown), and eventually causes NaN values.

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 /home/lbrown/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

5 Week of September 29th - October 1st

5.1 Goals for the Week

1. What I know from Last Week (kept here for ease of access)
2. Find out what causes some of the probabilties to go outrageously high
3. Compare Min, Max, and Average Values
4. Use Sections of the Simulated Yeast Genome
5. If possible, NSE Patch
6. More information on Codon Position
7. Look into an R Vim extension or RStudio
8. Debugging Thoughts

5.2 Progress/Notes

5.2.1 What I know from Last Week (kept here for ease of access)

Here's what I know about the NaN error

1. ~~one~~ One or more 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). In the fourth run, it happened in 3 and 8 (D and I). In the fifth, it was 4 (E).
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.

Cedric suggested that it may be caused by the covariance matrix exploding.

10. ~~All the values~~ The average 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.

11. The C code sometimes FIXES the problem! When doing a second run with seed 83455, lp.vec has a NaN value at position 22893 (amino acid???)

12. The Phi values for codons with -Inf log probability are absurdly high.

```
Browse[2]> mean(phi[is.finite(lp.c.raw)])
[1] 15240.47
Browse[2]> mean(phi[!is.finite(lp.c.raw)])
[1] 3.256039e+20
Browse[2]> testmat <- cbind(testvec[!is.finite(lp.c.raw)], lp.c.raw[!is.finite(lp.c.
Browse[2]> colnames(testmat) <- c("#", "lp.c.raw", "phi", "naa")

Browse[2]> testmat
        # lp.c.raw          phi naa
cnu   194     -Inf 1.041789e+07   4
csrD  226     -Inf 1.091396e+04  18
frmR  467     -Inf 1.756417e+08   4
malQ  829     -Inf 1.057376e+04  36
moaC  915     -Inf 1.814127e+05  10
mqsR  934     -Inf 4.044010e+06   7
rsxA 1433     -Inf 6.201766e+04   5
shoB 1485      NaN 4.558455e+21   2
tesB 1551     -Inf 1.629425e+05  10
ybfA 1738     -Inf 8.534363e+05   4
```

```
ycfJ 1801     -Inf 3.949059e+04  10
yciB 1818     -Inf 2.038530e+05  12
ygeR 1993     -Inf 6.640161e+04  20
yhaJ 2020     -Inf 2.098447e+05  14

Browse[2]> yaa[!is.finite(lp.c.raw),]
     AAA AAG
cnu    3   1
csrD  14   4
frmR   2   2
malQ  24  12
moaC   7   3
mqsR   5   2
rsxA   3   2
shoB   2   0
tesB   8   2
ybfA   3   1
ycfJ   7   3
yciB   5   7
ygeR  17   3
yhaJ  13   1
```

5.2.2 Find out what causes some of the probabilties to go outrageously high

Theories:

One value is going to infinity, for its own reasons. This causes the other values to drop in response, which explains the drop in the average lpProp values. Through some means (addition of the bias? Seems unlikely, logdmultinomCodAllR returns NaN values before the bias comes into effect), I think one of the probabilities goes above 1. Then the odds ratio $\frac{p_{\vec{c}_{ij}}}{1-p_{\vec{c}_{ij}}} > 1$, which causes the probabilites to EXPLODE. But, it stays under a cap, because stable_exp.c has the HUGE_VALUE loop (included above) that keeps things "under control". In attempting to scale the probabilities, it just slows down the code (because it's trying to scale down exponential growth by halves). Eventually, the value becomes so high that the C code cannot process it and returns NaN, which the R code refuses to use for the acceptance vector, and the code crashes.

Alternately, the other values could be going to 0, which causes certain values to become certain (or beyond certain) in response, as opposed to the converse.

To test this theory, I ran one crash test run of the NSE code (documented), with random seed 83455 and got a NaN number for lp.vec at amino acid 4, in lp.vec[7472]. I'm now rerunning the code with that random seed, and tracking the values of lp.vec[7472]. If I'm correct, it should eventually reach above 0(log probability > 0 means probability is > 1), and then skyrocket from there.

One other possibility is that it's some kind of numerical error e.g. underflow, and the value lp.vec[7472] will suddenly jump from -30ish to 1×10^{18}

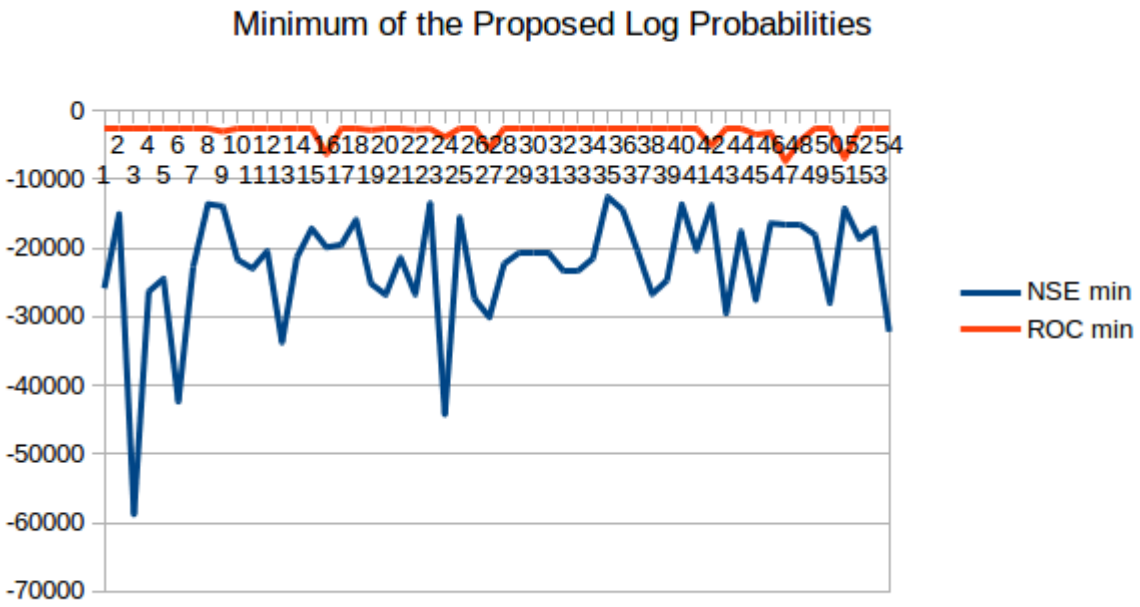
Results of test: Inconclusive

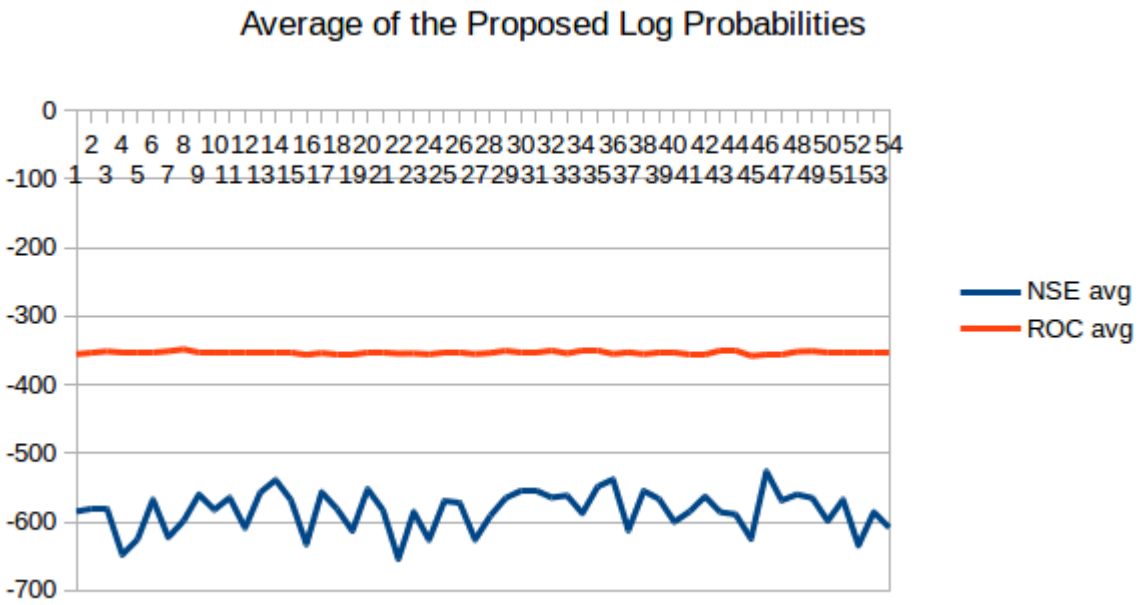
Setting the random seed to the same value generated DIFFERENT results. Cedric suggests that the MCMC may behave on a separate random seed. This is problematic.

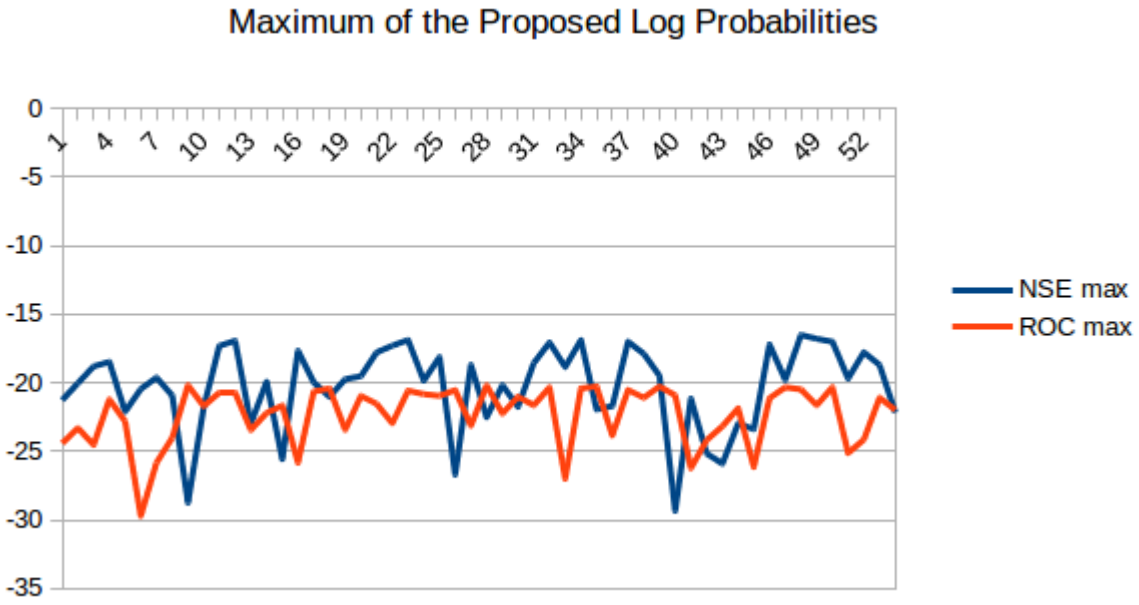
5.2.3 Compare Min, Max, and Average Values

grep (min/max/avg/NaN/inf) (file) | sort | (head/tail)

Wrote a small script, topbot.sh (min/max/avg/NaN/inf) (file)







5.2.4 Use Sections of the Simulated Yeast Genome

See cubfits/misc/S.cervisiae.REU13/random5ths.r

5.2.5 If possible, NSE Patch

One option to look into is to drop lp_c_raw from the code, and use rowsums like the Roc model. The two problems with that are that, as far as I can tell, lp.c.raw actually fixes the error in some cases, and using yaa * lp.vec complains for the NSE code. Not sure.

5.2.6 Look into an R Vim extension or RStudio

RStudio installed (also CMake), and it's VERY nice. Very intuitive, good use of screen real estate, and it holds a lot of information that I need at once. It keeps launching into /home/lbrown/PACKAGES/rstudio/bin though.

5.2.7 Debugging Thoughts

This was in last weeks summary, but it's included here for completeness.
How to GDB an R session

1. cd /home/lbrown/cubfits/misc/R
2. R -d gdb
3. run
4. source("debug_nsef.r")
5. Ctrl-C
6. continue

5.3 Goals for next Week

1. NSE Run Sections of Simulated Yeast Geome
2. Find out what causes some of the probabilties to go outrageously high
3. Compare Min, Max, and Average Values
4. Use Sections of the Simulated Yeast Genome
5. Look into an R Vim extension or RStudio
6. Debugging Thoughts