Work Log for December

Logan Brown

December 12, 2014

Contents

1 Goals for the Month 2

2 Progress/Notes 2
- 2.1 Verify logMu Flip by larger runs . . . 2
- 2.2 Investigate Problematic Codons . . . 5
- 2.3 Generate a new Genome . . . 10
  - 2.3.1 Fix the Code . . . 10
  - 2.3.2 Same inputs as Preston's Yeast . . . 10
  - 2.3.3 Preston's reference codon values + deltaOmega from CUBFits . . . 10
- 2.4 Is it worth it to adjust Delta a_12? . . . 10
- 2.5 Fix Names . . . 10
- 2.6 Speed up C code . . . 10
- 2.7 Move to Newton? . . . 11

3 Goals for next Month 11

1 Goals for the Month

As of December 1st.

1. Verify logMu Flip by larger runs
2. Investigate Problematic Codons
3. Is it worth it to adjust Delta a_12?
4. Fix Names
5. Move to Newton?

2 Progress/Notes

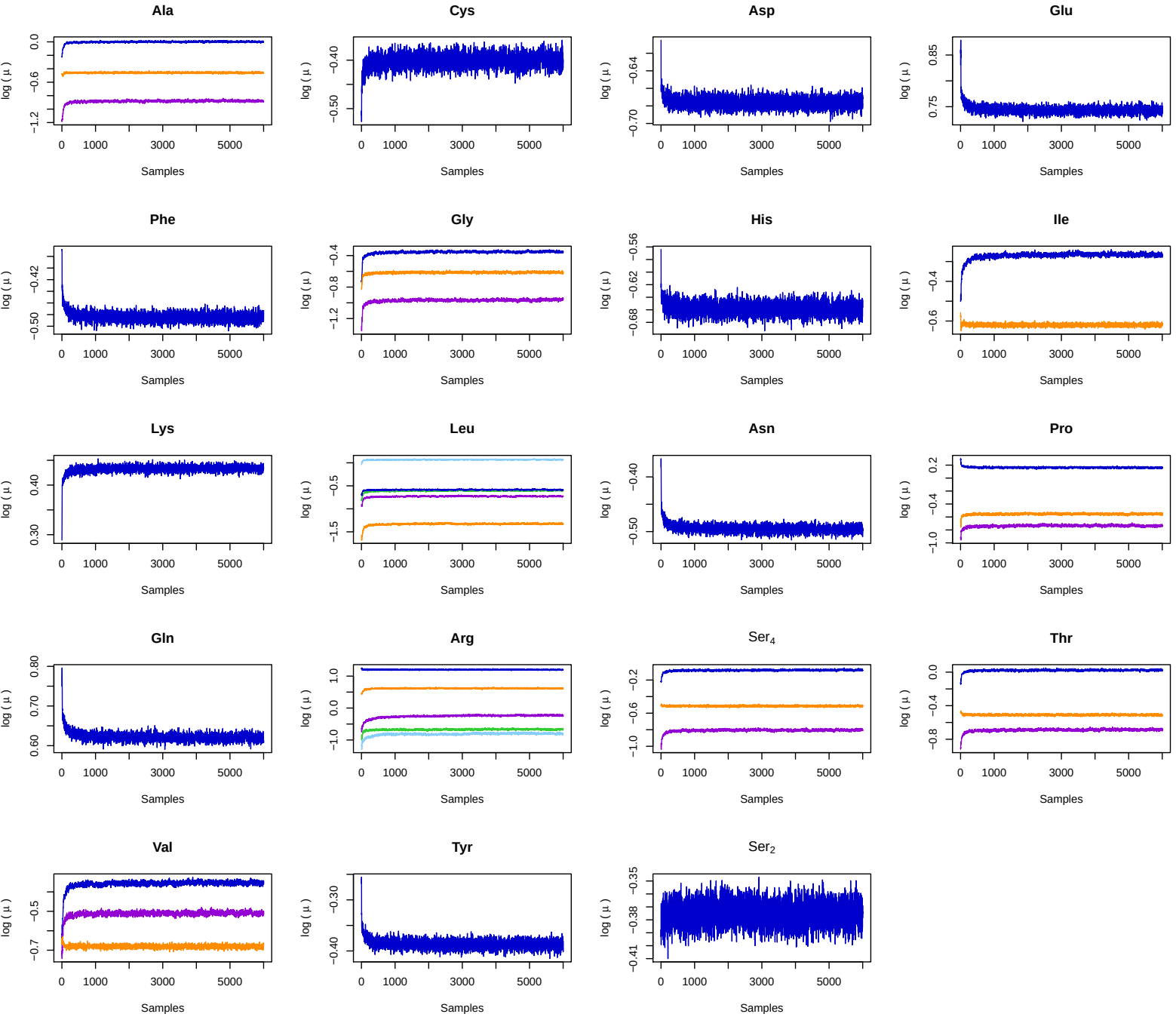
2.1 Verify logMu Flip by larger runs

Verified!

Doing a run of 6000 steps is definitely long enough, especially if you look at the log likelihood trace (not included for brevity). The logMu looks a bit more convincing, and also,it improve the behavior of the hyperparameters like σ_ϵ and σ_ϕ. For the first 300 or so samples, the model has to fit to the negative $log(\mu)$ value. To its credit, the model does so, but that's not good.

AA parameter trace 11−21

Mon Dec 1 13:15:18 2014



AA parameter trace 11−24

Mon Dec 1 13:14:01 2014

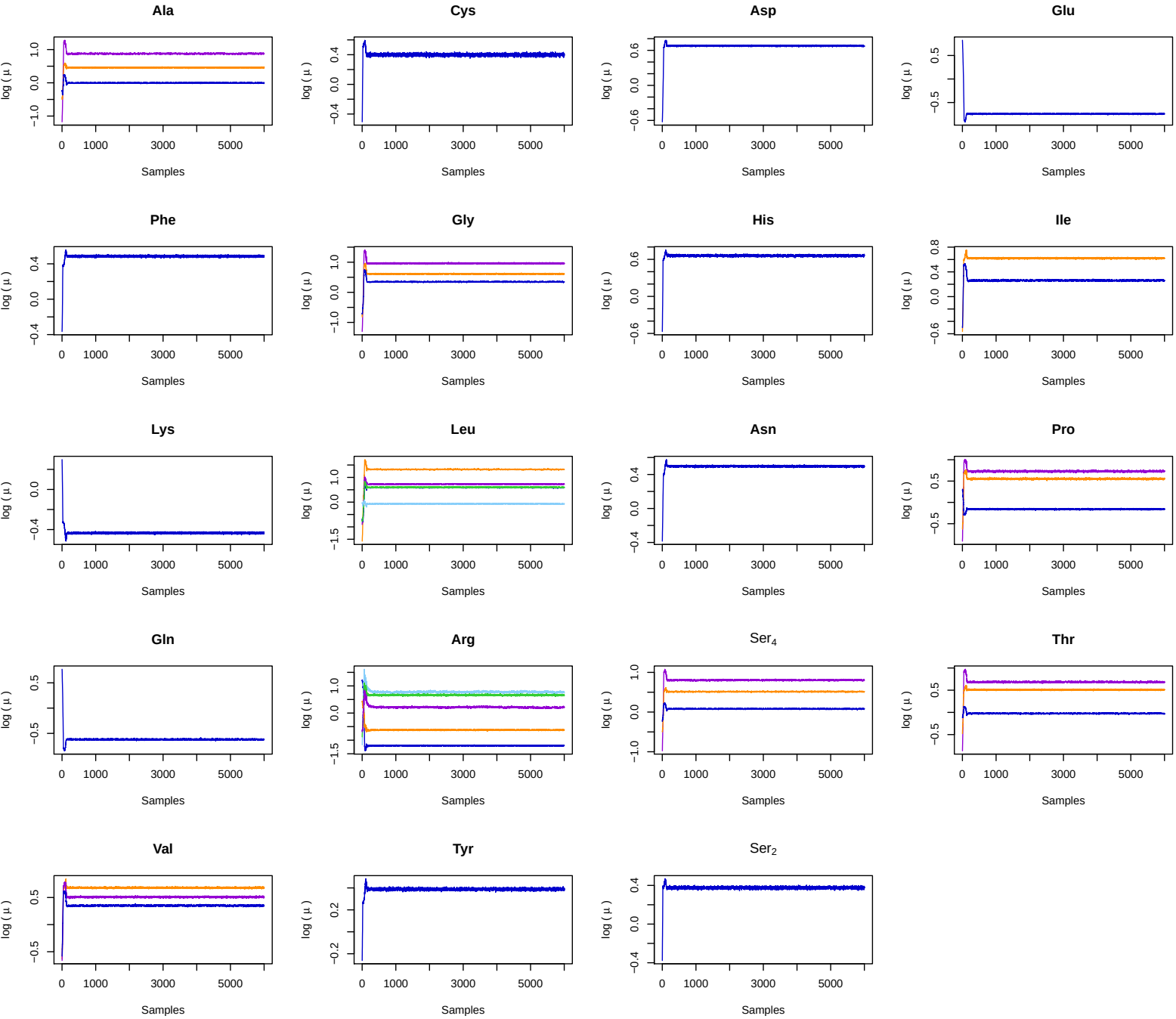
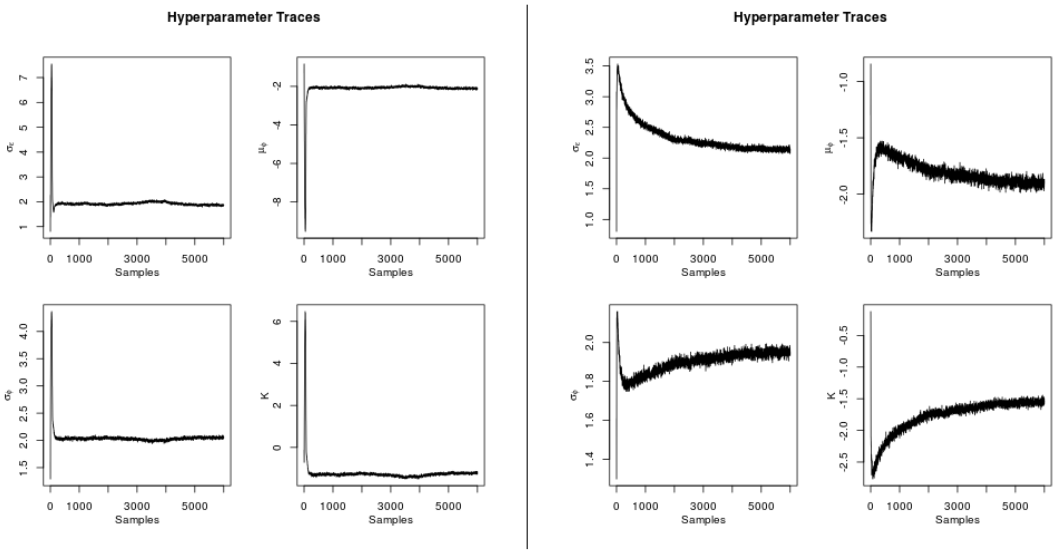


Figure 1: The left figure is a run before the logMu flip. The right is after. You can see that before fixing the logMu flip, the hyperparameters see a huge spike early on, which quickly subsides, while on the right, you simply see them increase at the beginning as the model begins to fit, then fall off slowly as the model converges.



2.2 Investigate Problematic Codons

One thing we were interested in looking at was comparing the problematic ω values to their problematic logμ values. Here's a mapping!

Green Square: Leucine CTT

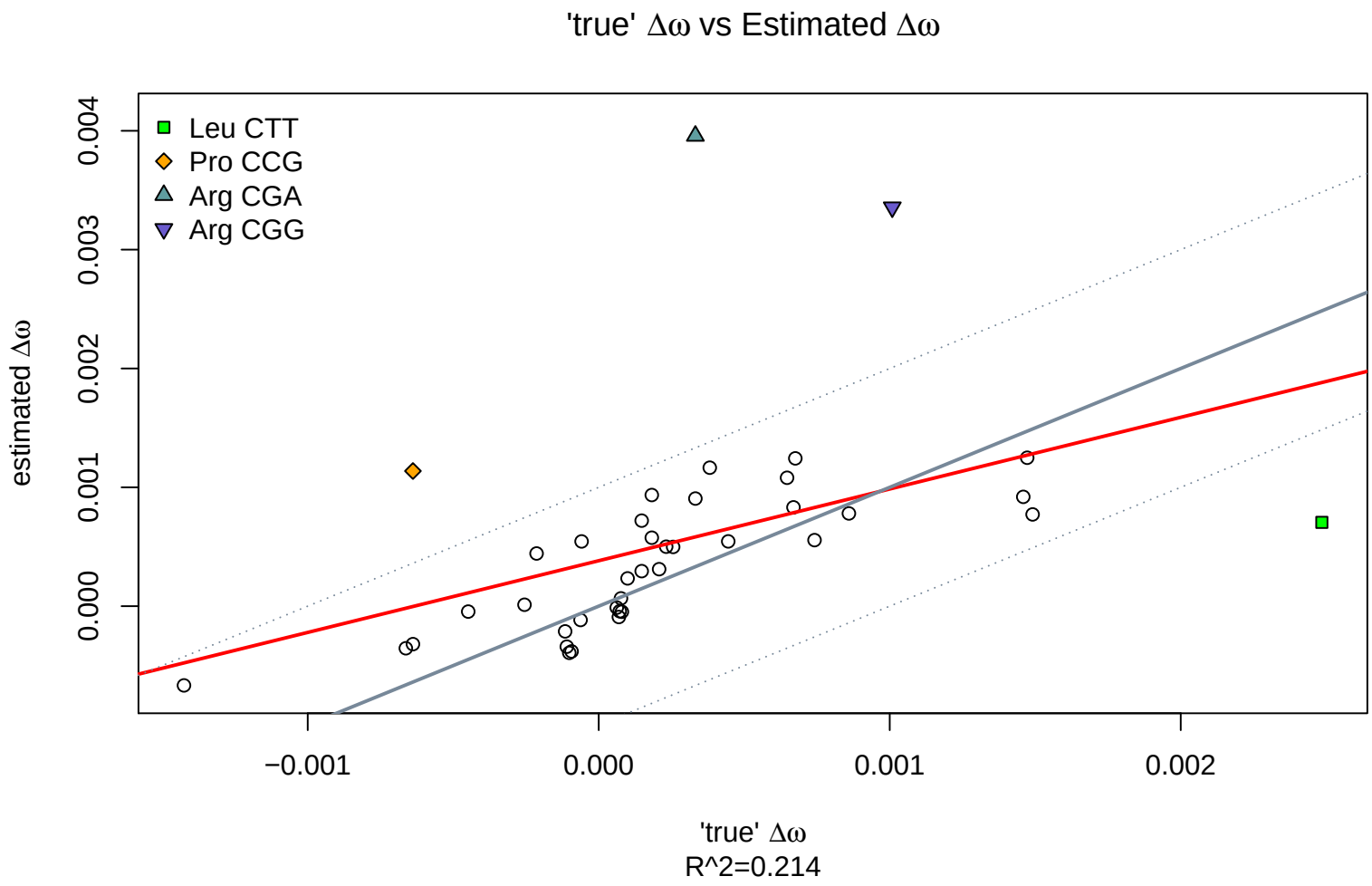
Yellow Diamond: Proline CCG

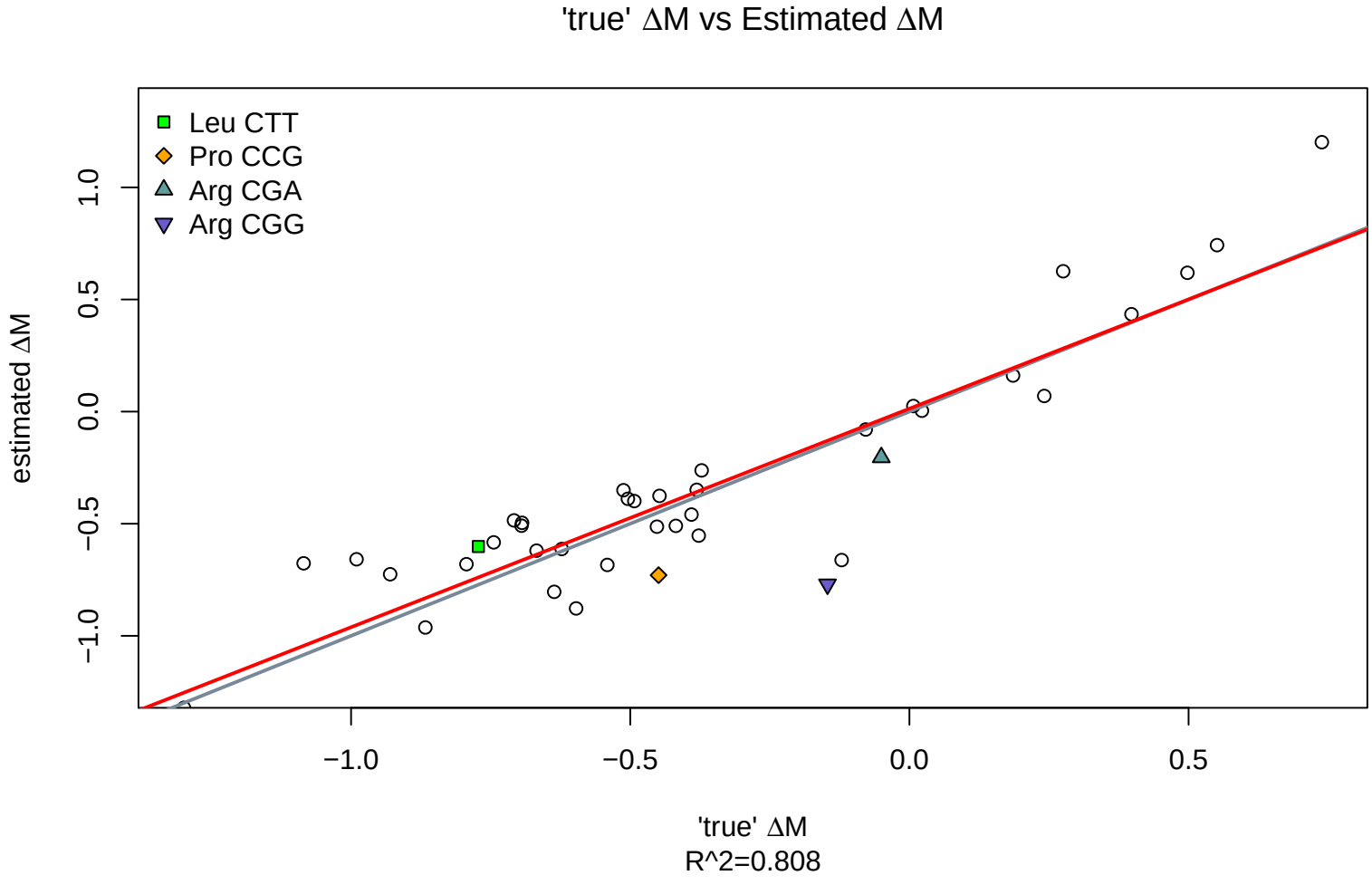
Blue Up Arrow: Arginine CGA

Purple Down Arrow: Arginine CCG

As we anticipated, higher nonsense error rates lead to lower mutation rates, and lower nonsense error rates lead to higher mutation rates. The magnitude is a bit off, the lowest mutation does not cause the highest nonsense errors.

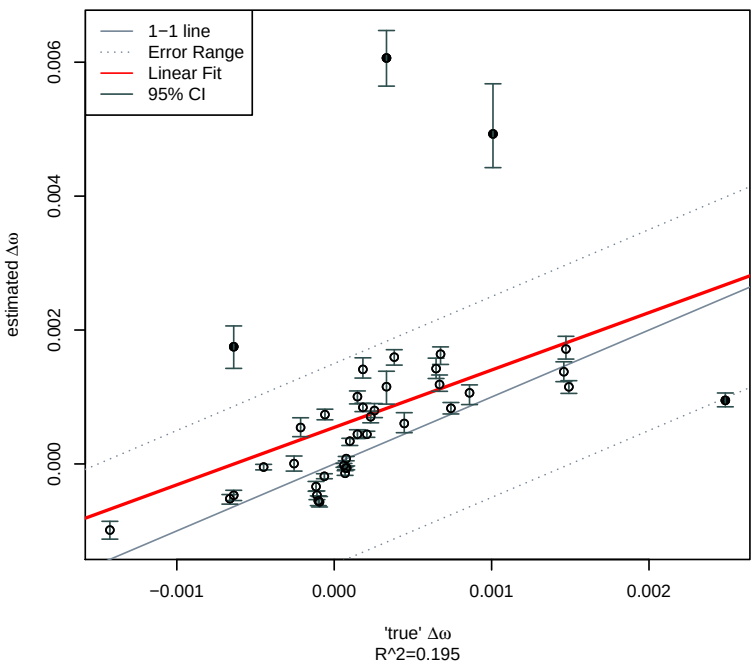
This was also reproduced when using different sections of the Yeast Genome. Here are 3 distinct sections of preston's simulated yeast (they share no genes in common) that produce similar results.


'true' Δω vs Estimated Δω
Leu CTT
Pro CCG
Arg CGA
Arg CGG
estimated Δω
0.000
0.001
0.002
0.003
0.004
−0.001
0.000
0.001
0.002
'true' Δω
R^2=0.214

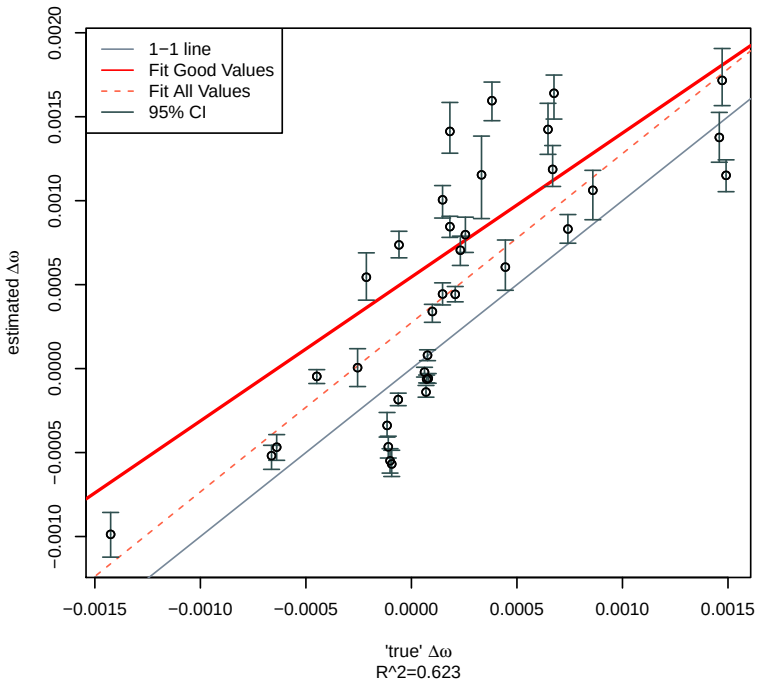

'true' ΔM vs Estimated ΔM
Leu CTT
Pro CCG
Arg CGA
Arg CGG
estimated ΔM
−1.0
−0.5
0.0
0.5
1.0
−1.0
−0.5
0.0
0.5
'true' ΔM
R^2=0.808

Section 1

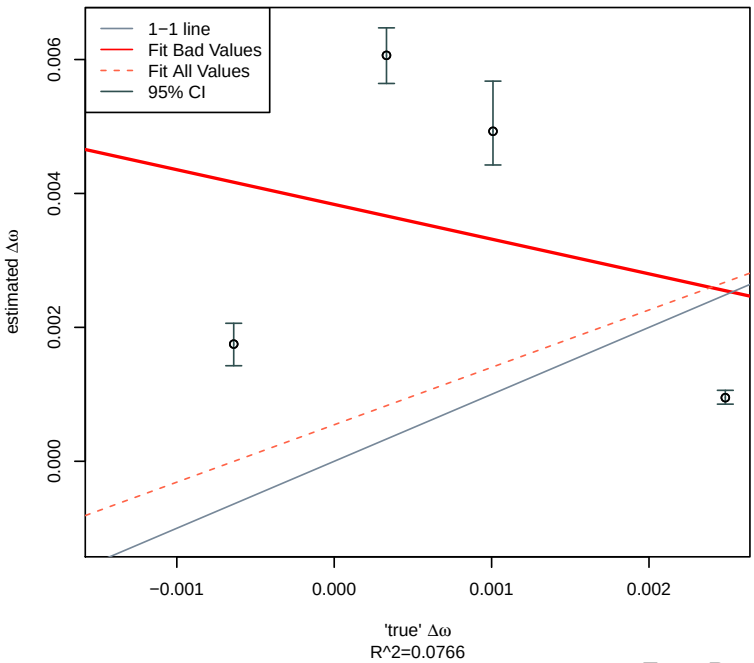
raw error 'true' Δω vs estimated Δω

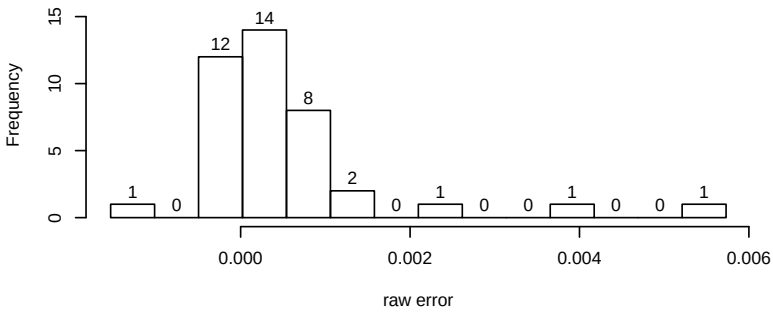


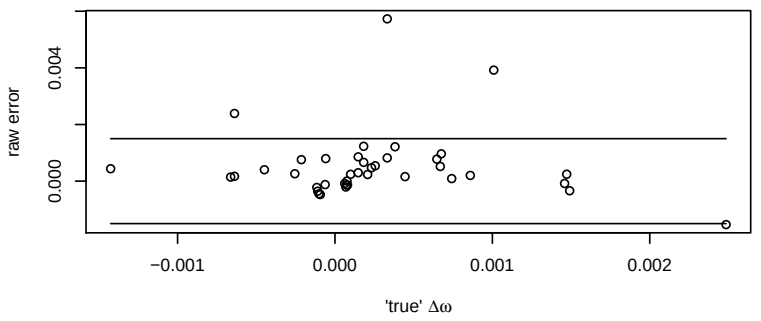
raw error Δω without problem values



raw error Δω problem values



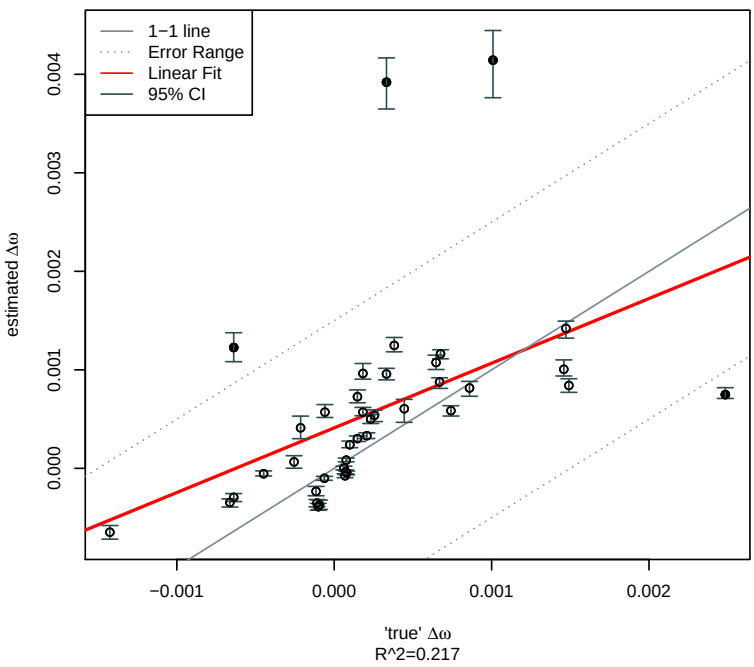




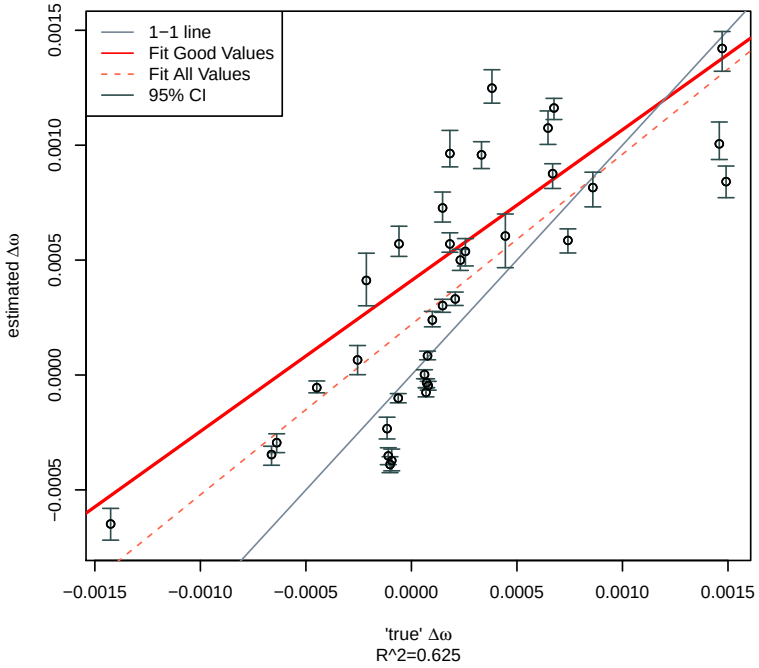
Error Range is +/− 0.0015

Section 3

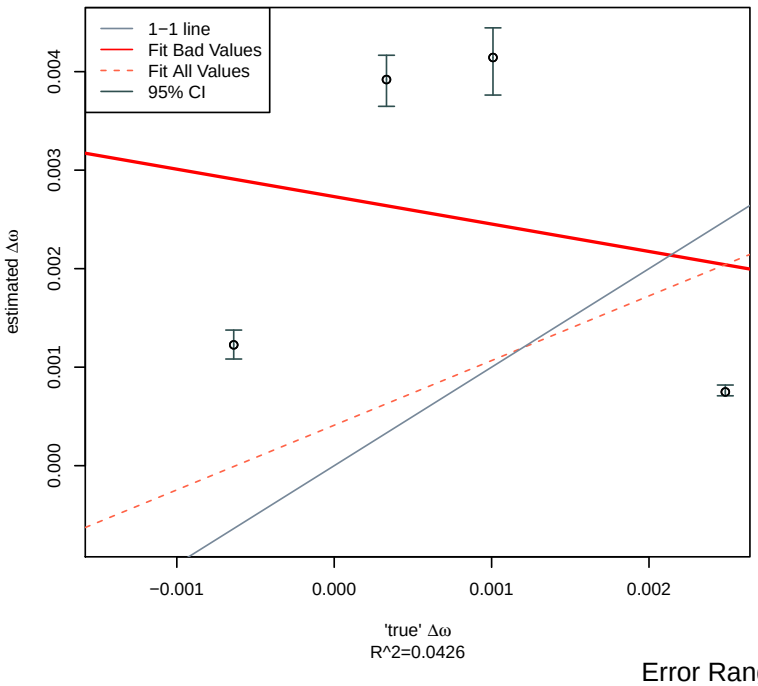
raw error 'true' Δω vs estimated Δω

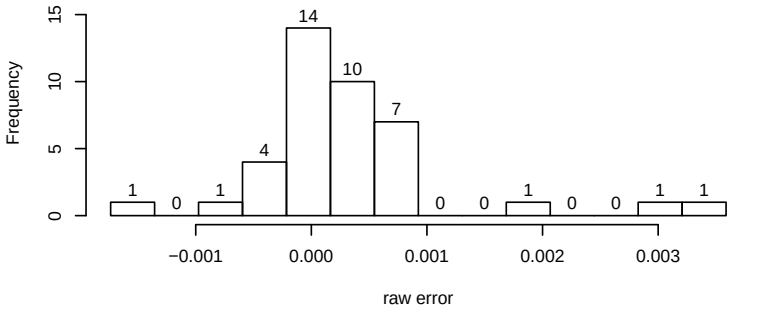


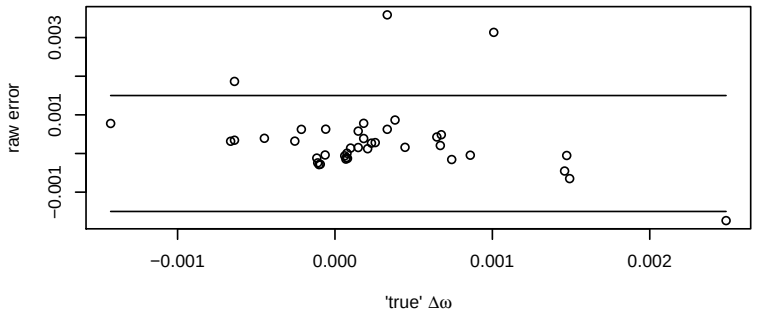
raw error Δω without problem values



raw error Δω problem values



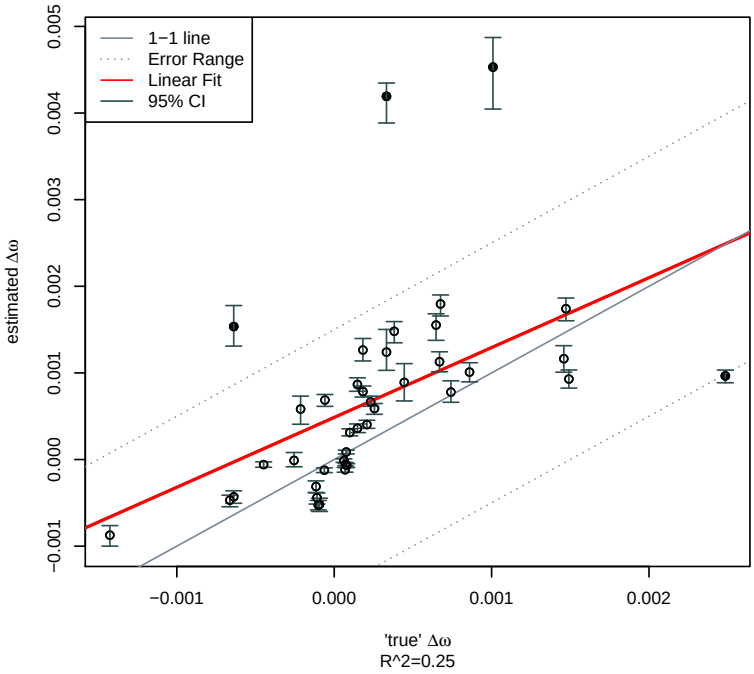




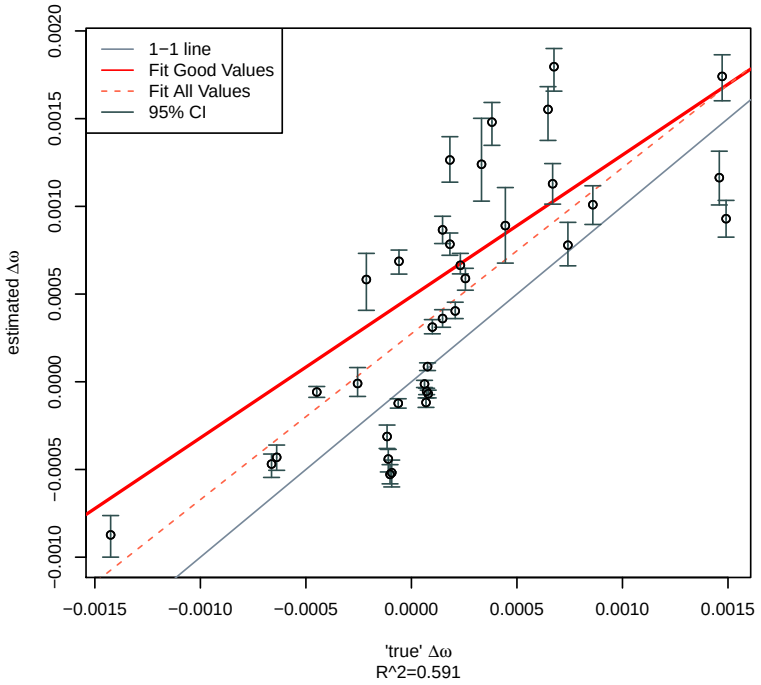
Error Range is +/− 0.0015

Section 4

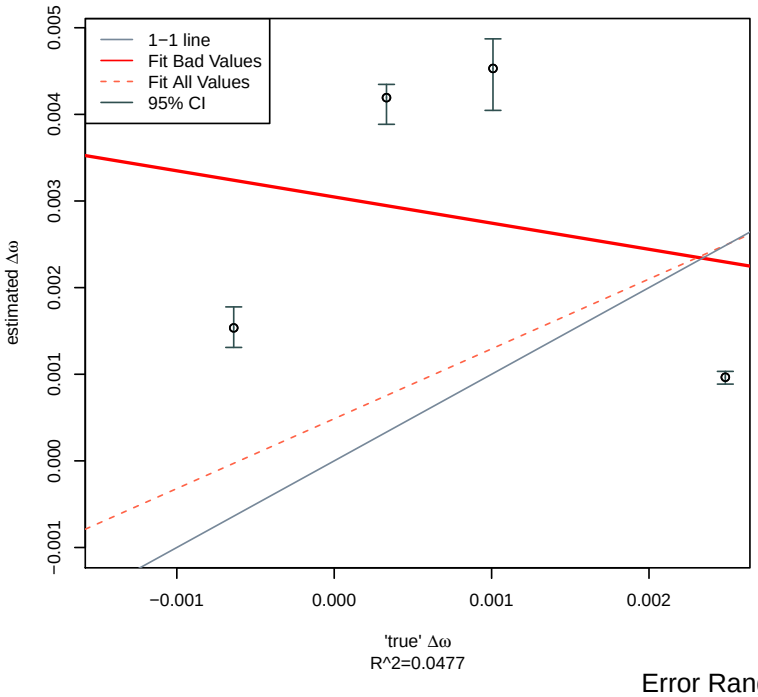
raw error 'true' Δω vs estimated Δω

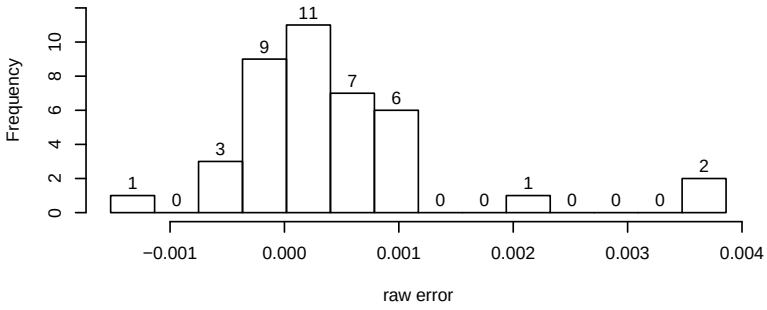


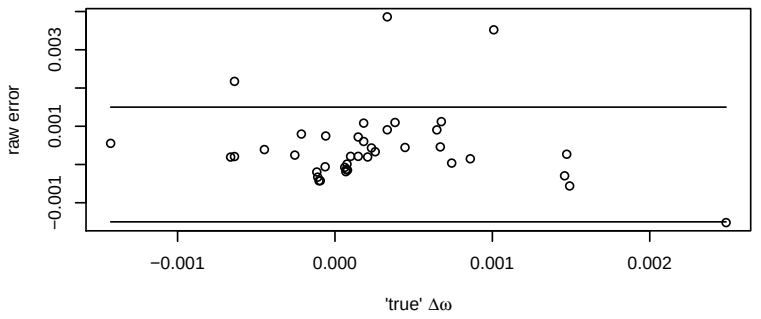
raw error Δω without problem values



raw error Δω problem values







Error Range is +/− 0.0015

2.3 Generate a new Genome

DONE.

~~By modifying preston's old code, I was able to create two new simulated genomes that used the same inputs as Preston's yeast, but are new and different.~~

2.3.1 Fix the Code

Preston's code had a problem where the simulation was only updating the c_index, but was never actually updating the codons used. Then, when writing the genome, it wrote out the codons used. The result was that the output was identical to the input, even when the input was nonsense.

My fix uses a for loop to generate a character array of the correct codons, which it then converts to a factor and uses to replace the one in gene.list$gene.dat$codon

If I work on this again in the future, I want to change the code so that the stop codon is no longer given index 99, so that I can actually mutate and translate the stop codon over

These changes are also being committed to github in a repository forked from clandere.

2.3.2 Same inputs as Preston's Yeast

2.3.3 Preston's reference codon values + deltaOmega from CUBFits

2.4 Is it worth it to adjust Delta a_12?

2.5 Fix Names

May have fixed the names by doing the following in cedric.mapBMatNames.r

```
if(model == "roc"){
```

to

```
if(model == "roc" || model == "nsef"){
```

Luckily, the functions already exist to get the coefficients for the ROC, NSE and ROCandNSE models, so I didn't have to change anything (so far)

2.6 Speed up C code

I made the following change in stable_exp.c

```
for(k = 0; k < *K; k++){
       a_Z_normalized[k] -= max_exp;
}
```

```
for(k = 0; k < *K; k++){
        a_Z_normalized[k] = exp(a_Z_normalized[k]);
        *total_sum += a_Z_normalized[k];
}
```

to

```
for(k = 0; k < *K; k++){
        a_Z_normalized[k] = exp(a_Z_normalized[k] - max_exp);
        *total_sum += a_Z_normalized[k];
}
```

The actual time on these operations is quite small, but since it has to happen ~~hundreds of thousands of times~~ 2.8 million times for each step, it may actually cause an impact. Who knows?

The actual time difference is hard to measure, apparently.

I can measure it in microseconds, but the difference appears to be less than a microsecond, because it tends to come out as either a difference of 1 or 0 microseconds. There is a strange error where in the old version, the difference is 20 or more microseconds. If we give those credit, I seem to have reduced the run by between 1.4 and 5 microseconds. This doesn't sound like a lot, however, this code gets run millions of times for each cubfits run. Simulated yeast has 2.8 million codons, which means that this reduction would be 3.9 seconds less per proposal. Assuming 50,000 proposal states, that's 54 hours less. Which is quite unbelievable, since the runs don't even go that long.

So, assuming distances greater than about 5 microseconds are unbelievable, we can extrapolate what the actual change is. By averaging the values less than 5 microseconds (mostly 0s and 1s), we can guess how much the change actually is in microseconds. Using conservative estimates, it looks to be about .05 microseconds. Assuming 2.8 million codons and 50,000 MCMC proposed states, that's still a difference of 2 hours per run. The actual difference could be closer to .1 microseconds, which would be 4 hours per run.

These changes have been committed to my cubfits github, and a pull request to GilchristLab.

2.7 Move to Newton?

3 Goals for next Month

1. Future Goal