Work Log for November

Logan Brown

November 19, 2014

Contents

1 Goals for the Month

As of October 31st

1. Use Preston's Simulated Yeast, compare to REU yeast

   look for estimated $\approx$ 4*true

2. Parallelize the Code

   mclapply, getOption("mc.cores")?

3. Wei Chen's Yeast / Real Yeast Genome

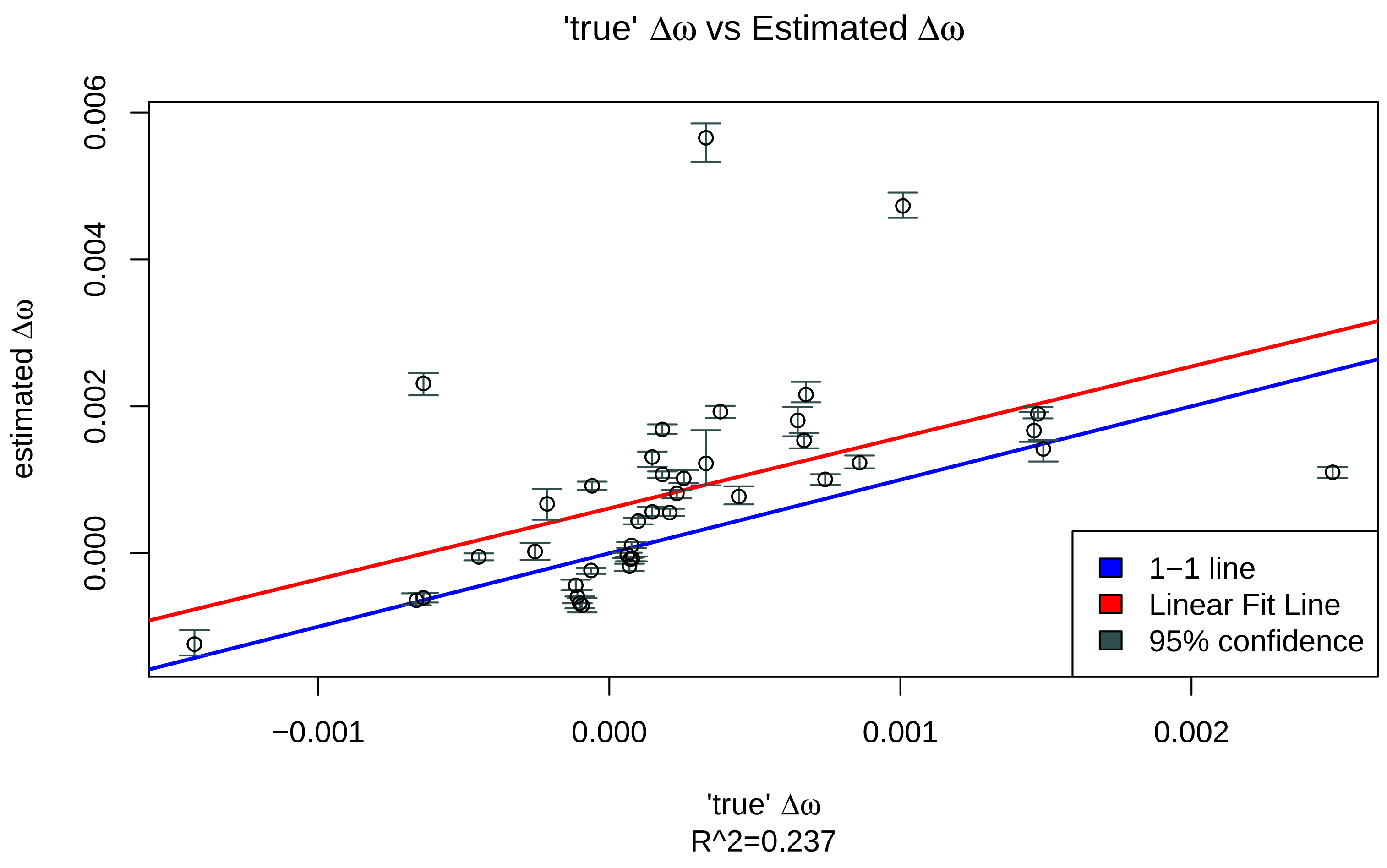
4. Generate my own simulated yeast, using a reverse engineered cubfits

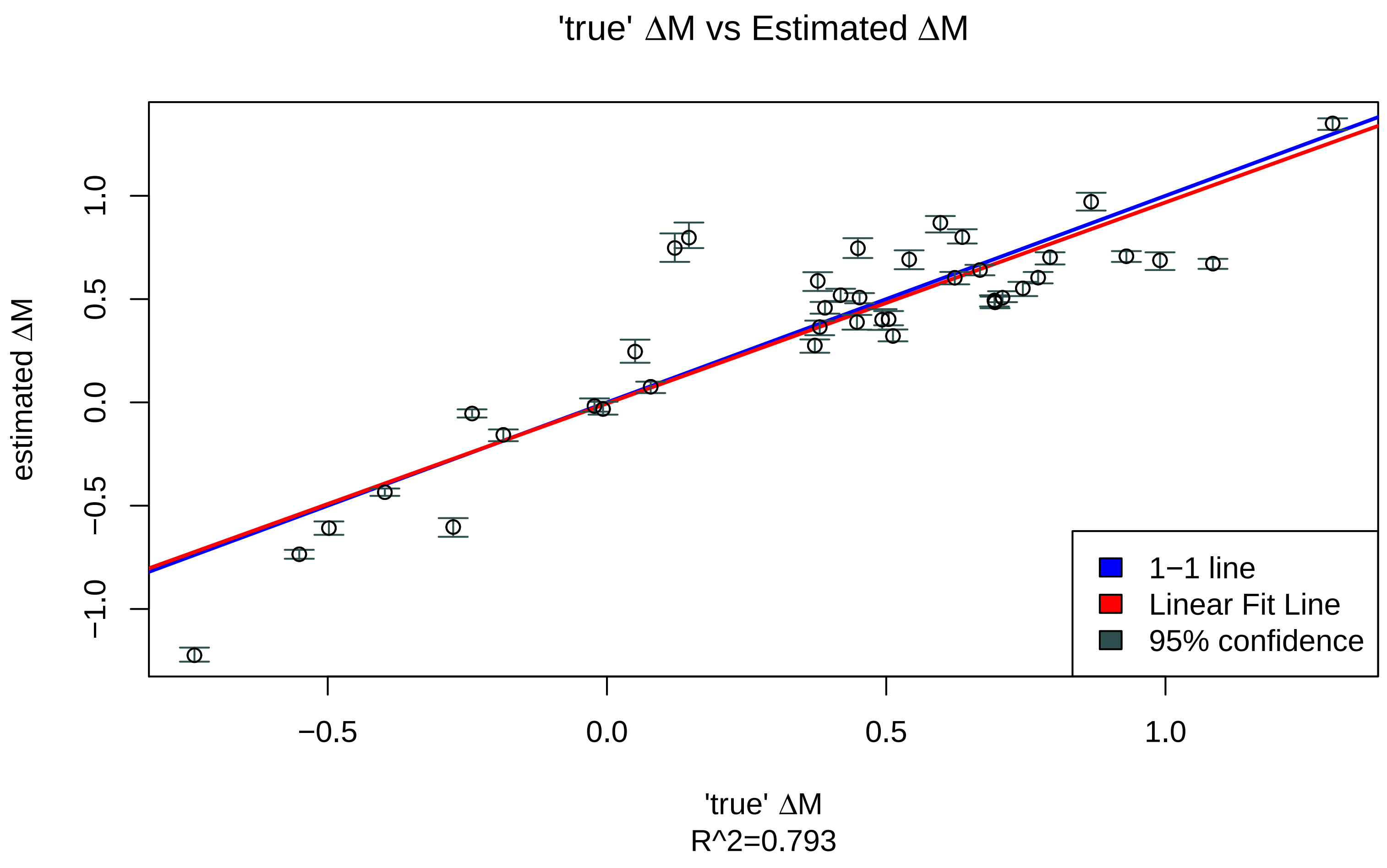
2 Progress/Notes

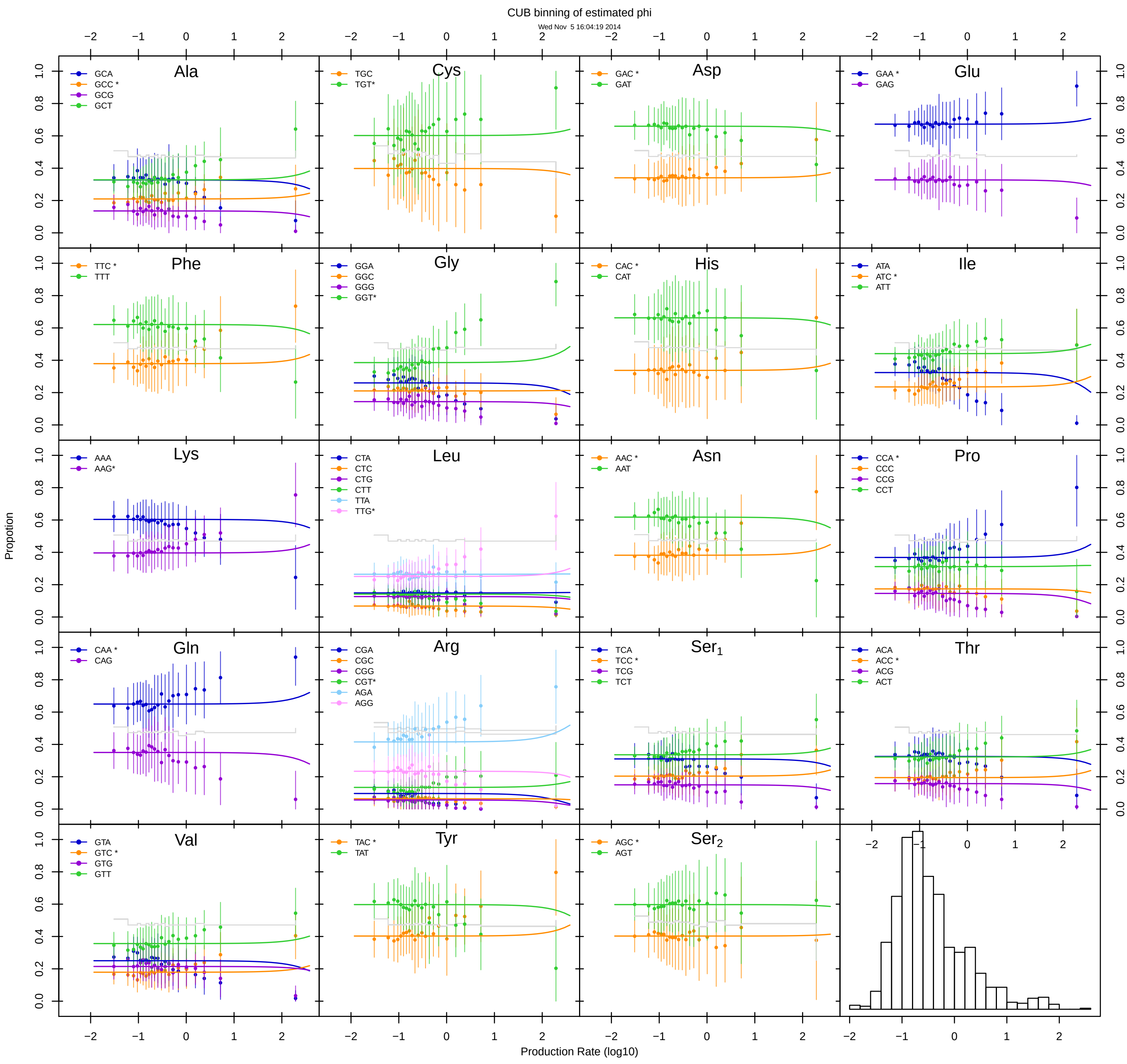
2.1 Use Preston's Simulated Yeast, compare to REU yeast

Here are the results from Preston's simulated yeast...

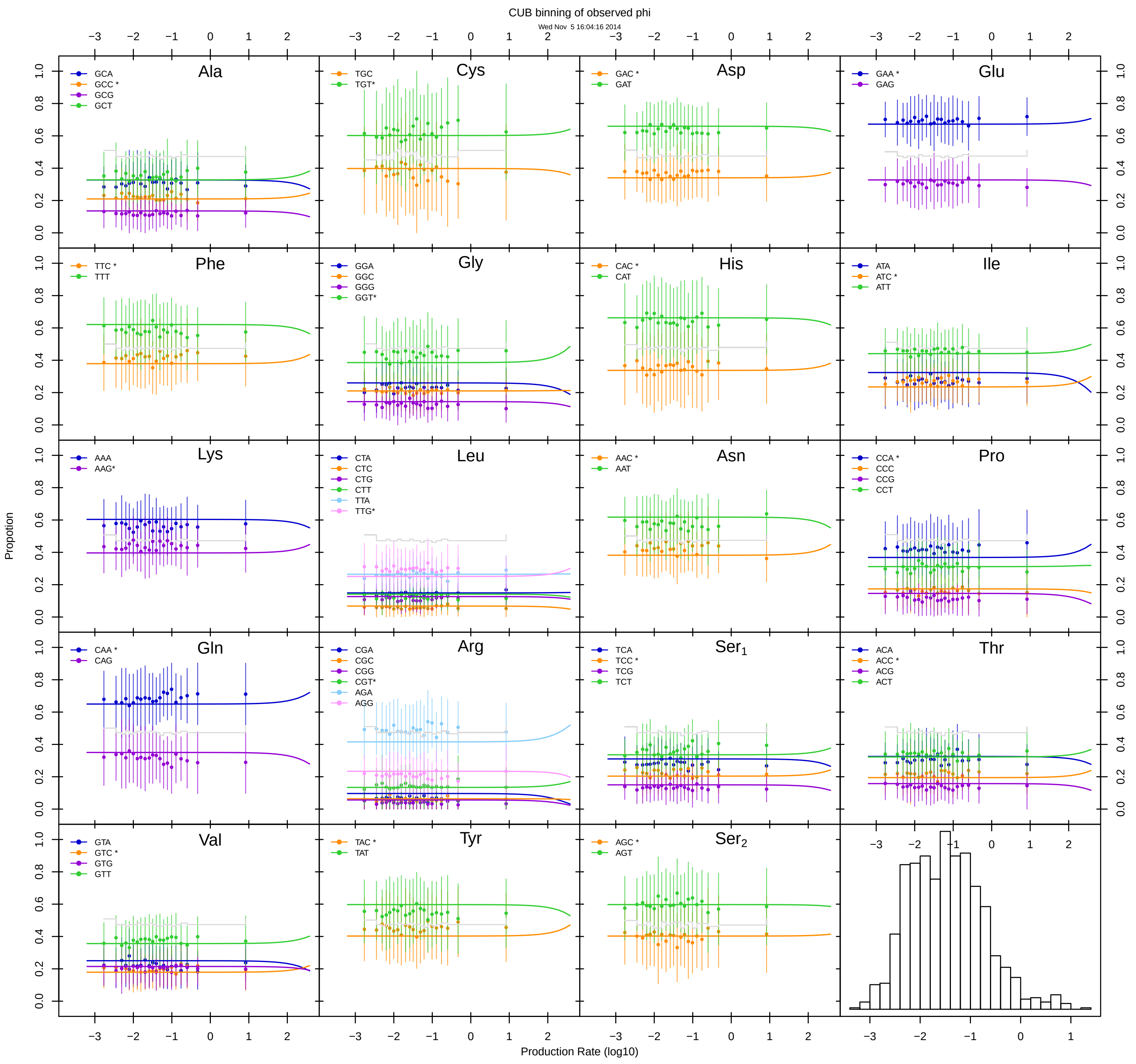
Notice that the $\Delta\omega$ values are not off by that $\approx$ 4 factor. We got a pretty good correlation on the logμ values, but the ϕ is pretty lousy, and the ω values leave something to be desired.


'true' Δω vs Estimated Δω
estimated Δω
0.000
0.002
0.004
0.006
−0.001
0.000
0.001
0.002
'true' Δω
R^2=0.237
1−1 line
Linear Fit Line
95% confidence


'true' ΔM vs Estimated ΔM
estimated ΔM
−1.0
−0.5
0.0
0.5
1.0
−0.5
0.0
0.5
1.0
'true' ΔM
R^2=0.793
1−1 line
Linear Fit Line
95% confidence



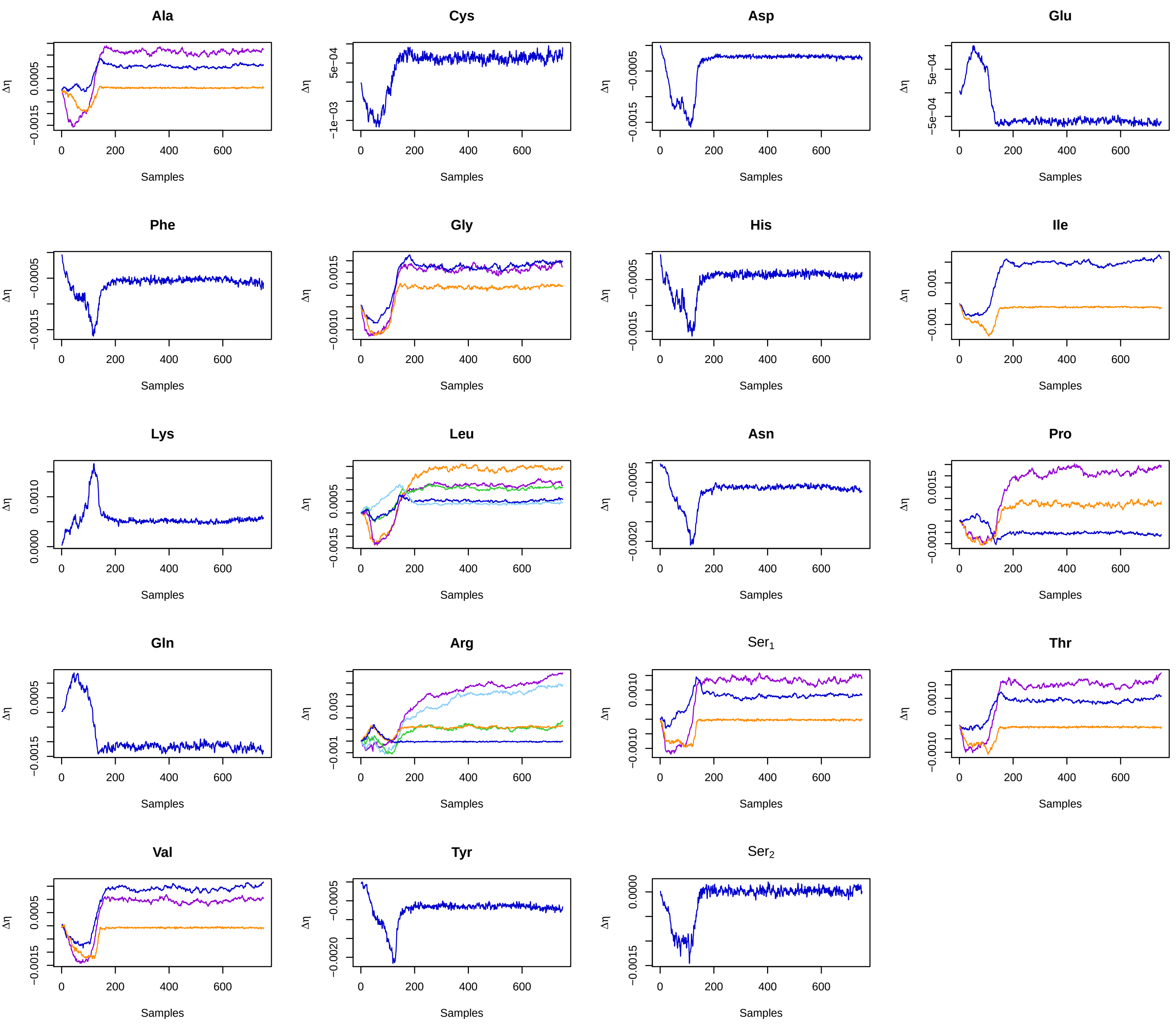
CUB binning of estimated phi
Wed Nov 5 16:04:19 2014
Ala
GCA
GCC *
GCG
GCT
Cys
TGC
TGT*
Asp
GAC *
GAT
Glu
GAA *
GAG
Phe
TTC *
TTT
Gly
GGA
GGC
GGG
GGT*
His
CAC *
CAT
Ile
ATA
ATC *
ATT
Lys
AAA
AAG*
Leu
CTA
CTC
CTG
CTT
TTA
TTG*
Asn
AAC *
AAT
Pro
CCA *
CCC
CCG
CCT
Gln
CAA *
CAG
Arg
CGA
CGC
CGG
CGT*
AGA
AGG
Ser1
TCA
TCC *
TCG
TCT
Thr
ACA
ACC *
ACG
ACT
Val
GTA
GTC *
GTG
GTT
Tyr
TAC *
TAT
Ser2
AGC *
AGT
Propotion
Production Rate (log10)

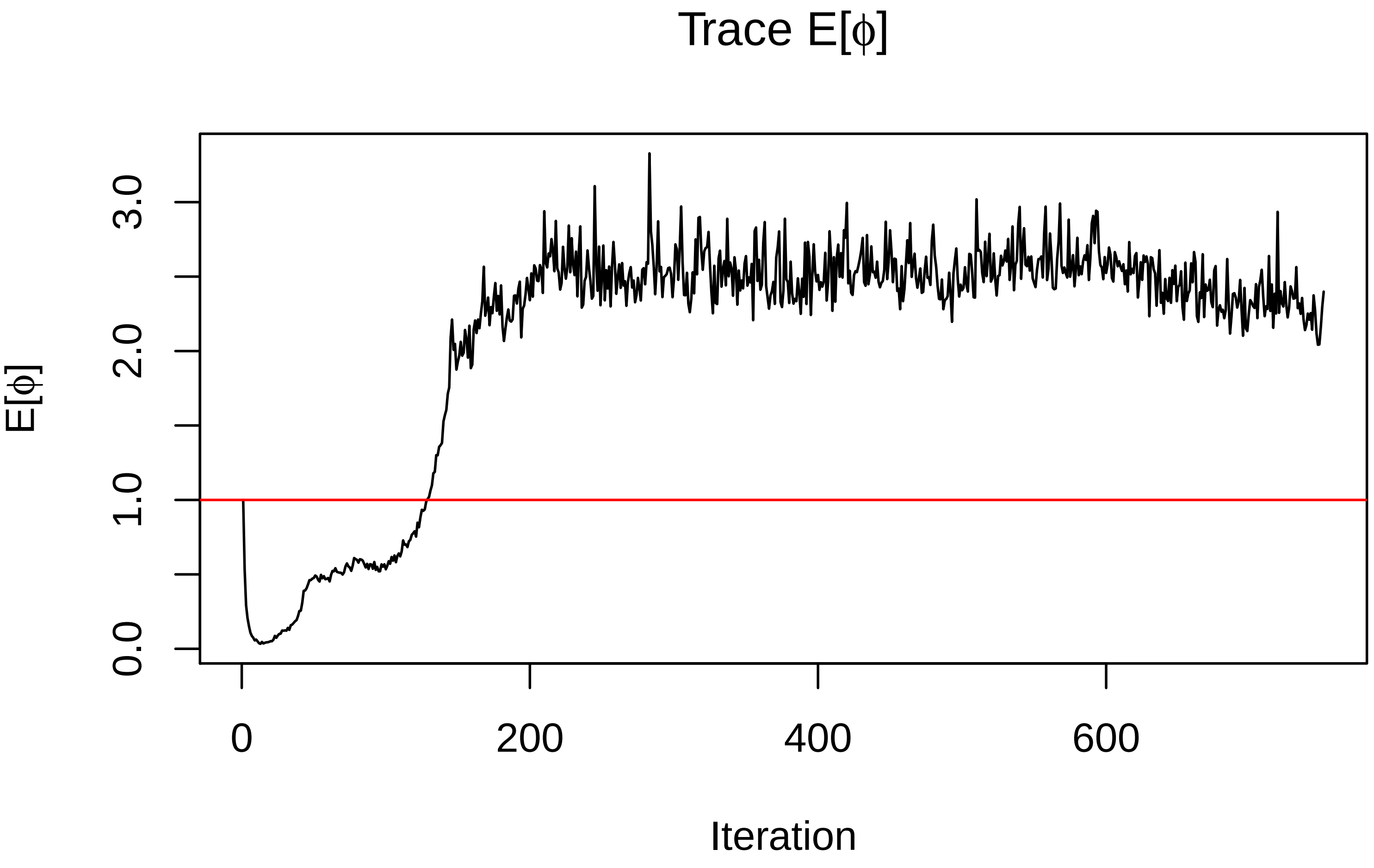


CUB binning of observed phi
Wed Nov 5 16:04:16 2014
Ala
GCA
GCC *
GCG
GCT
Cys
TGC
TGT*
Asp
GAC *
GAT
Glu
GAA *
GAG
Phe
TTC *
TTT
Gly
GGA
GGC
GGG
GGT*
His
CAC *
CAT
Ile
ATA
ATC *
ATT
Lys
AAA
AAG*
Leu
CTA
CTC
CTG
CTT
TTA
TTG*
Asn
AAC *
AAT
Pro
CCA *
CCC
CCG
CCT
Gln
CAA *
CAG
Arg
CGA
CGC
CGG
CGT*
AGA
AGG
Ser1
TCA
TCC *
TCG
TCT
Thr
ACA
ACC *
ACG
ACT
Val
GTA
GTC *
GTG
GTT
Tyr
TAC *
TAT
Ser2
AGC *
AGT
Propotion
Production Rate (log10)

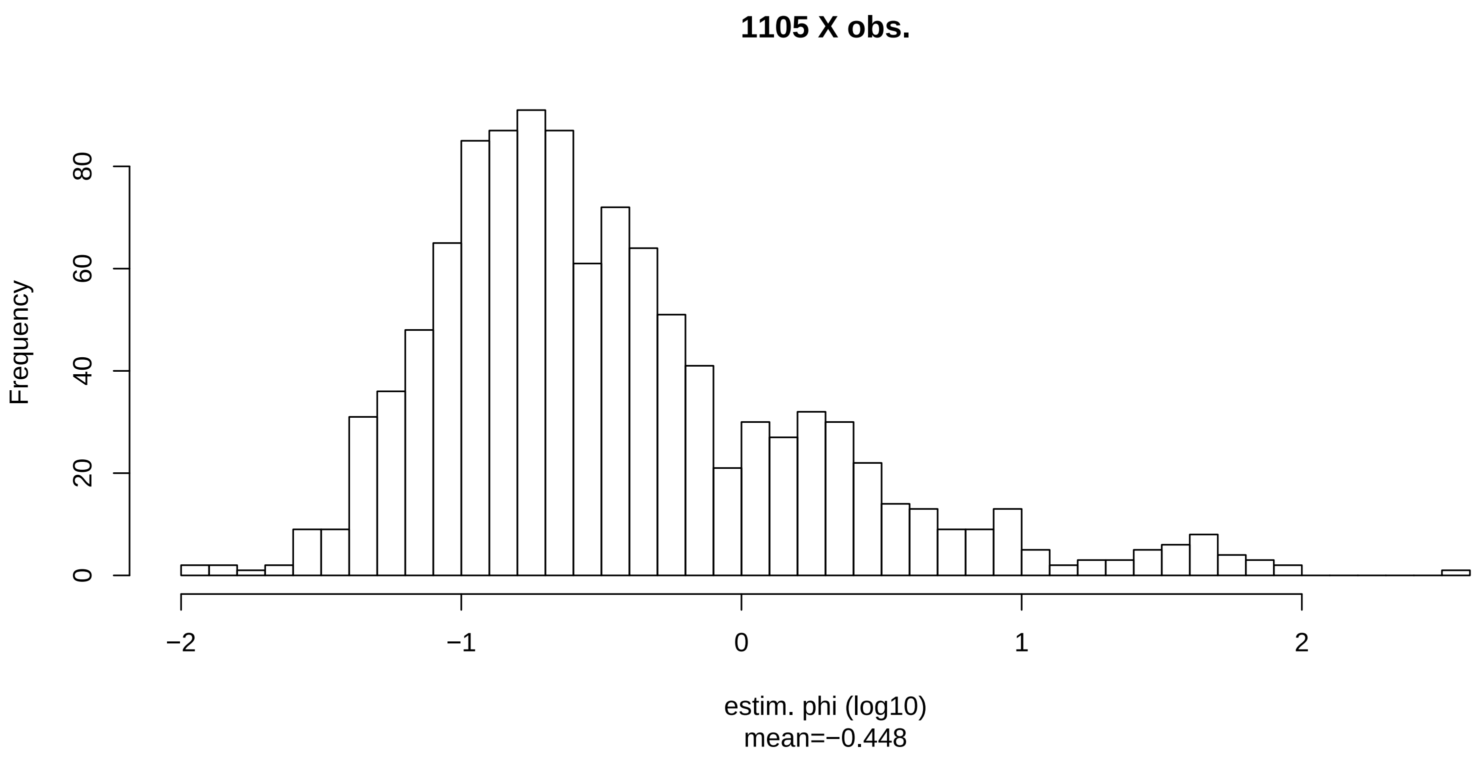
AA parameter trace 1105

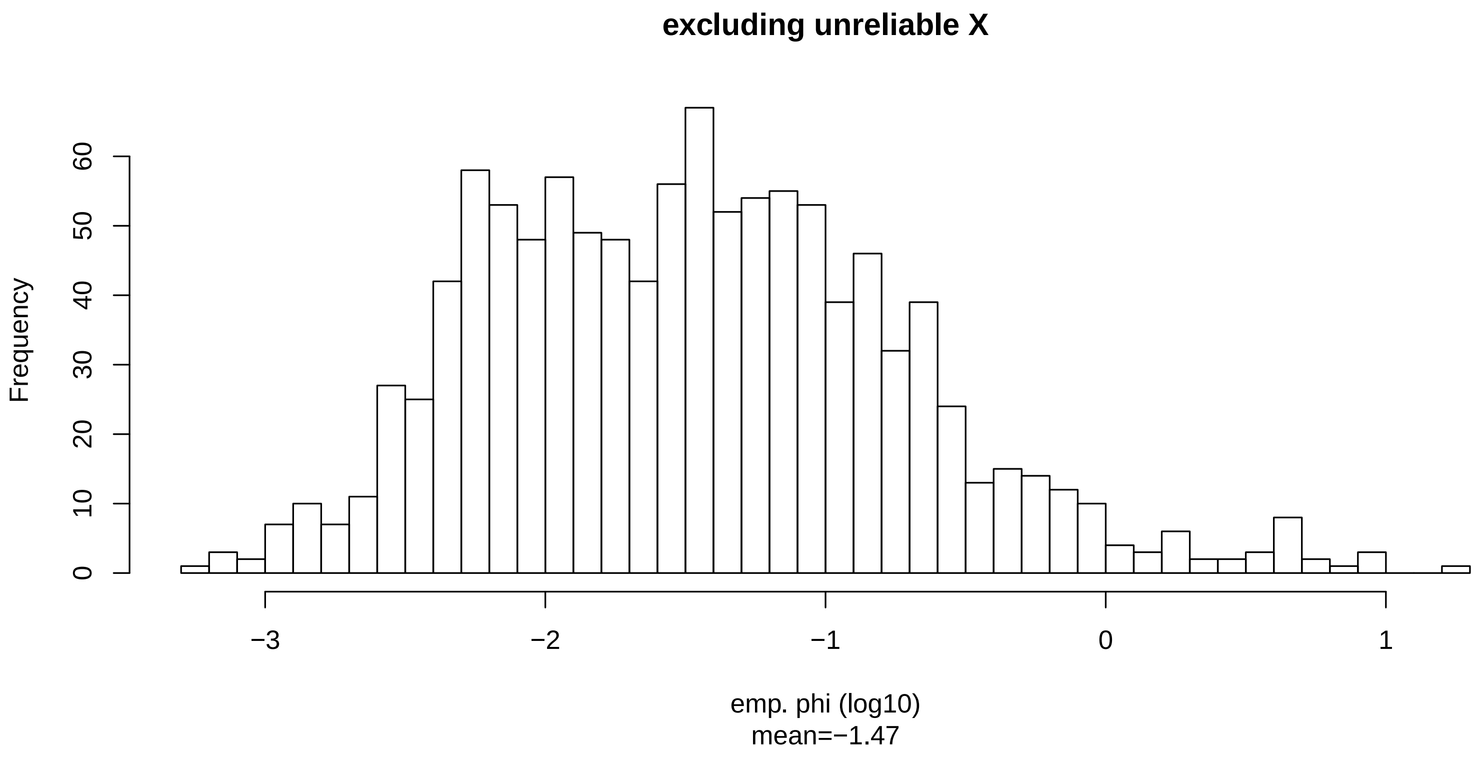
Wed Nov 5 16:04:14 2014



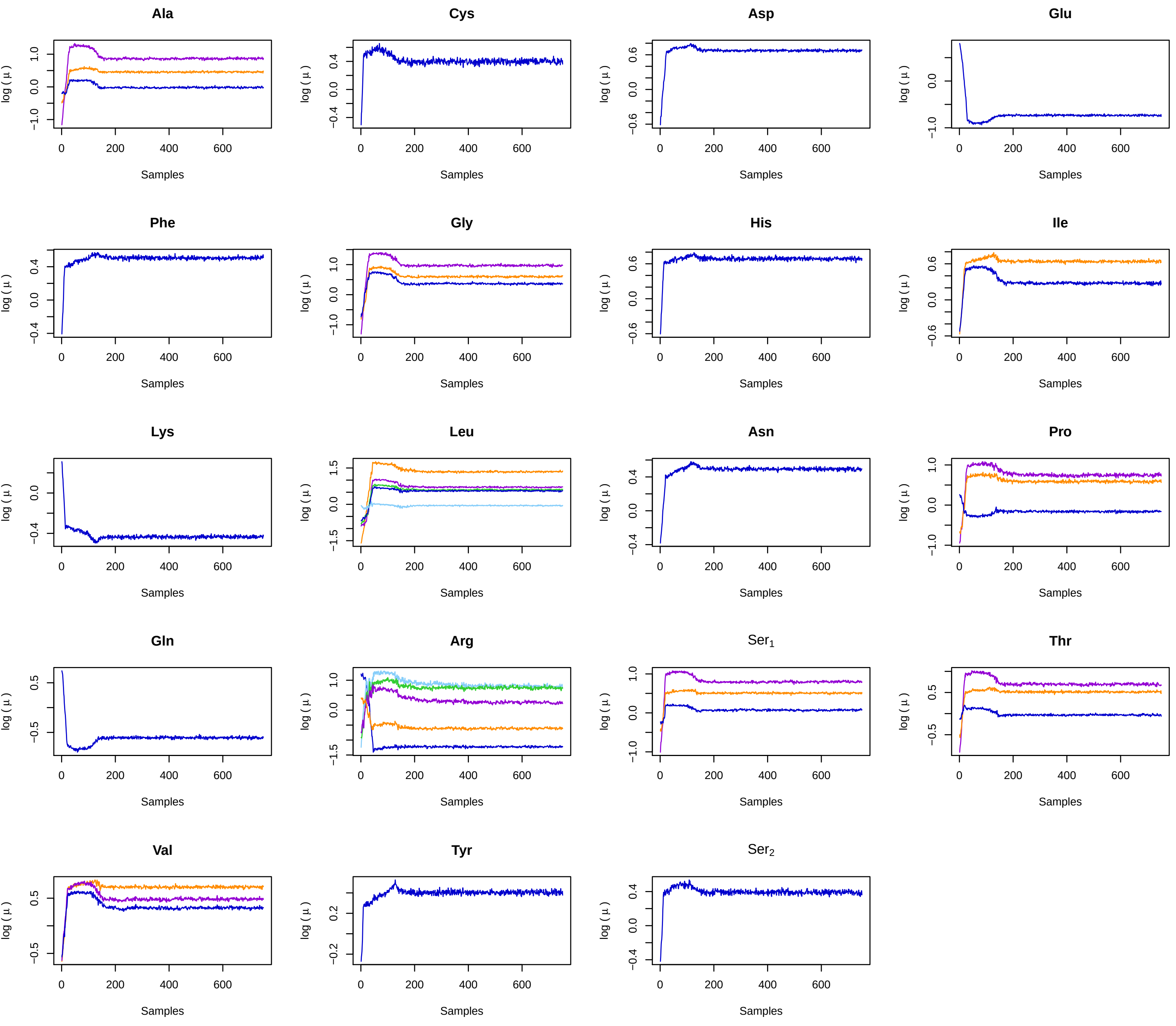


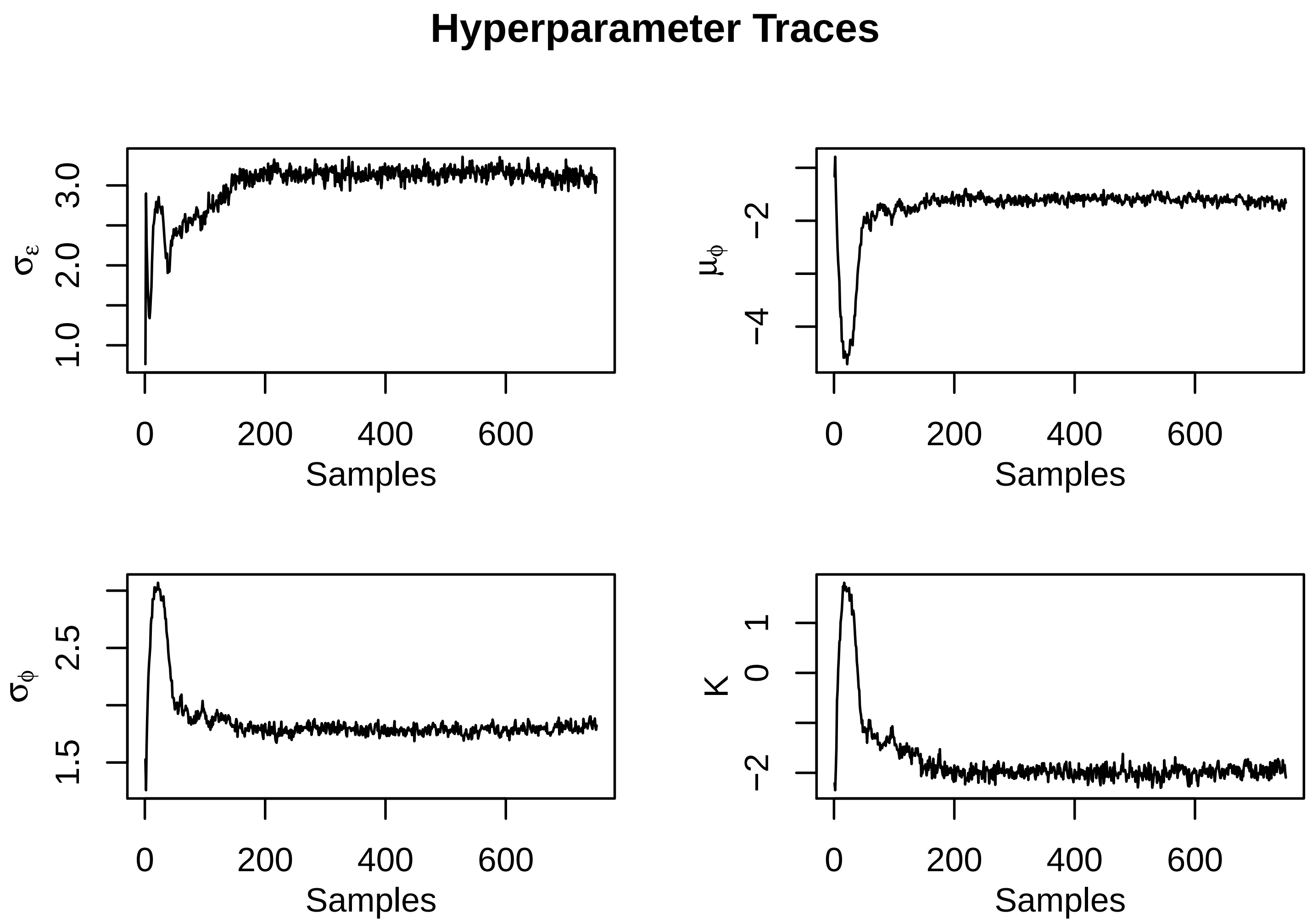
Trace E[ϕ]
E[ϕ]
0.0
1.0
2.0
3.0
0
200
400
600
Iteration


1105 X obs.
Frequency
0
20
40
60
80
−2
−1
0
1
2
estim. phi (log10)
mean=−0.448

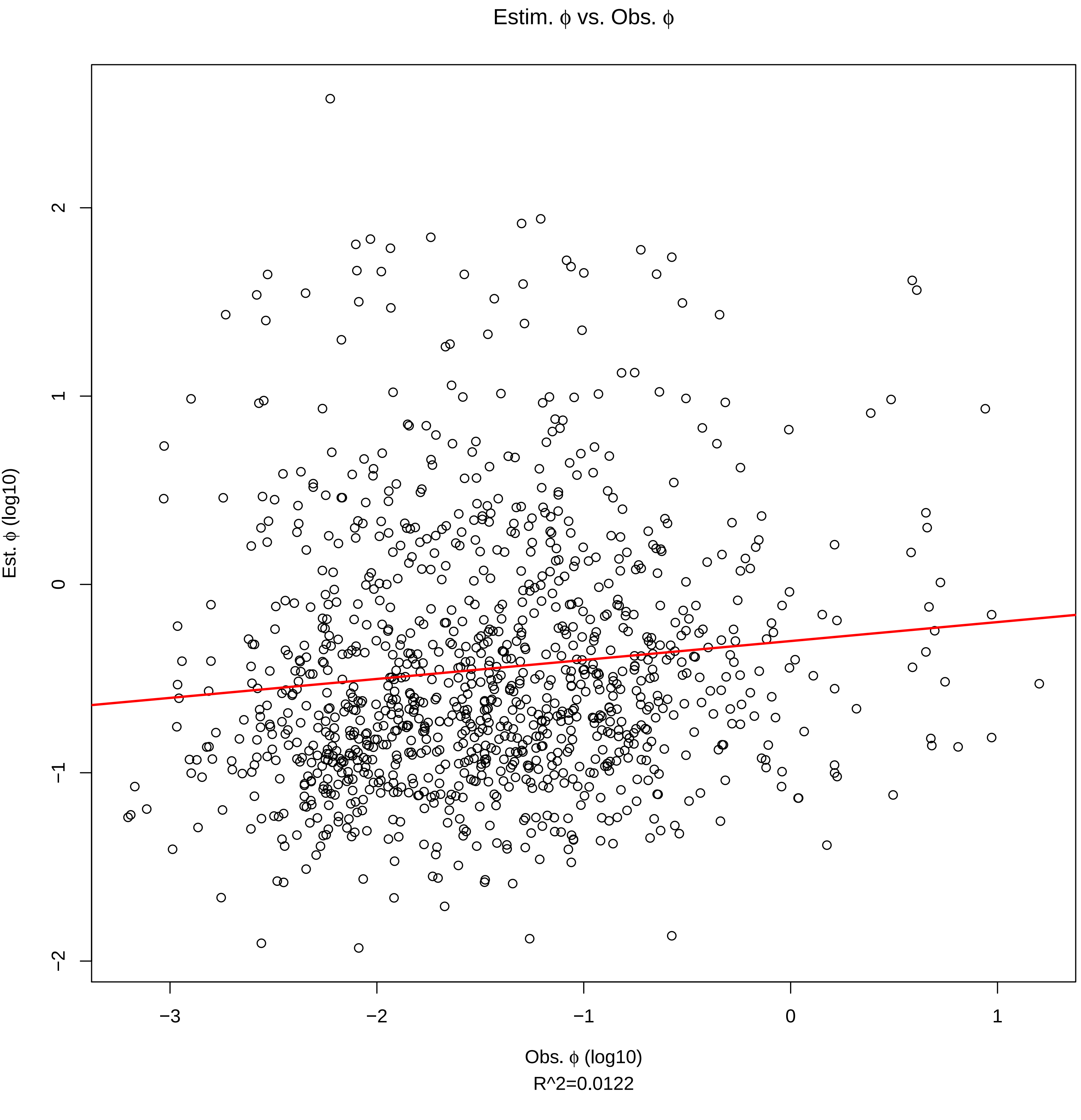

excluding unreliable X
Frequency
0
10
20
30
40
50
60
−3
−2
−1
0
1
emp. phi (log10)
mean=−1.47

AA parameter trace 1105
Wed Nov 5 16:04:13 2014



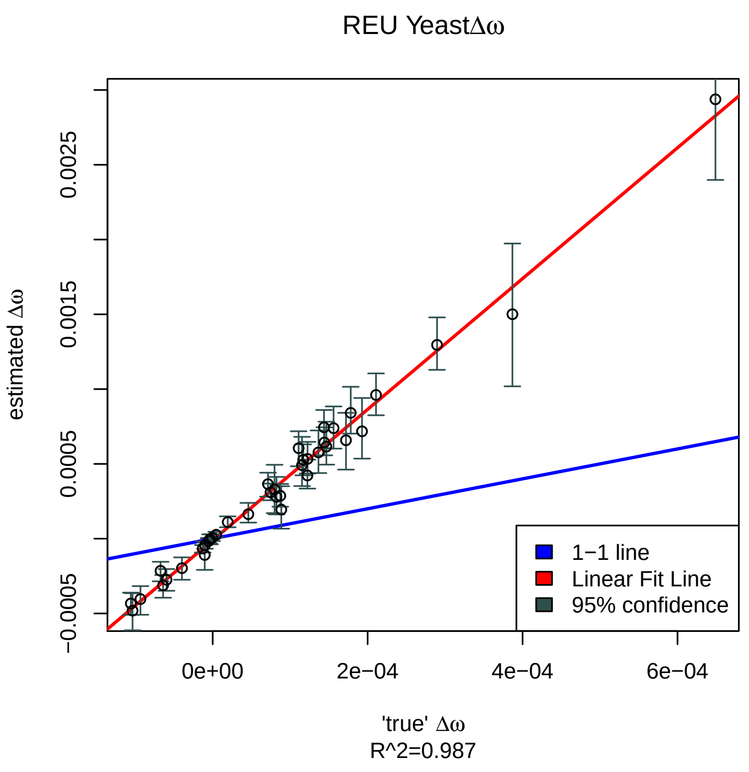


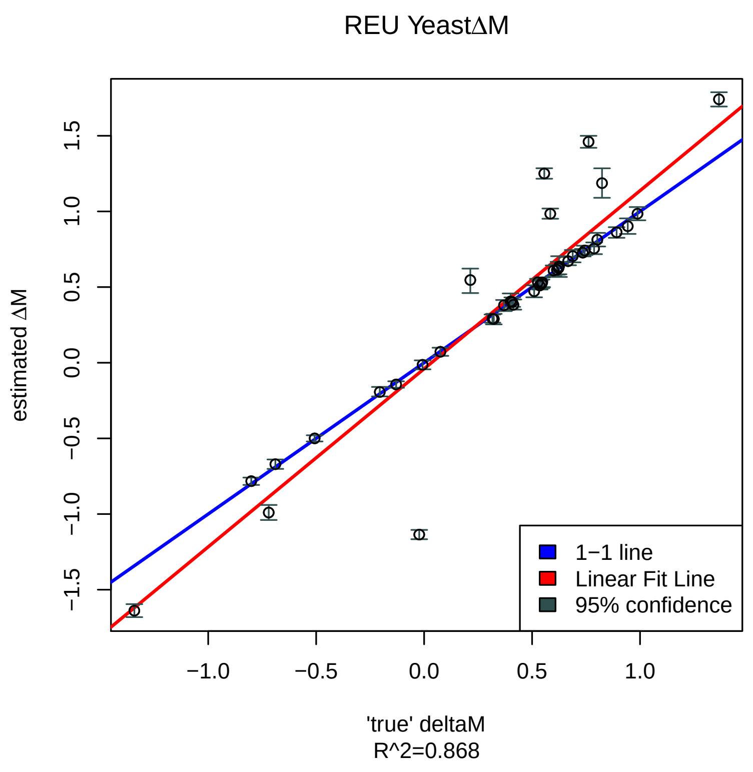
Hyperparameter Traces
σ_ε
3.0
2.0
1.0
μ_ϕ
−2
−4
σ_ϕ
2.5
1.5
K
1
0
−2
0
200
400
600
Samples

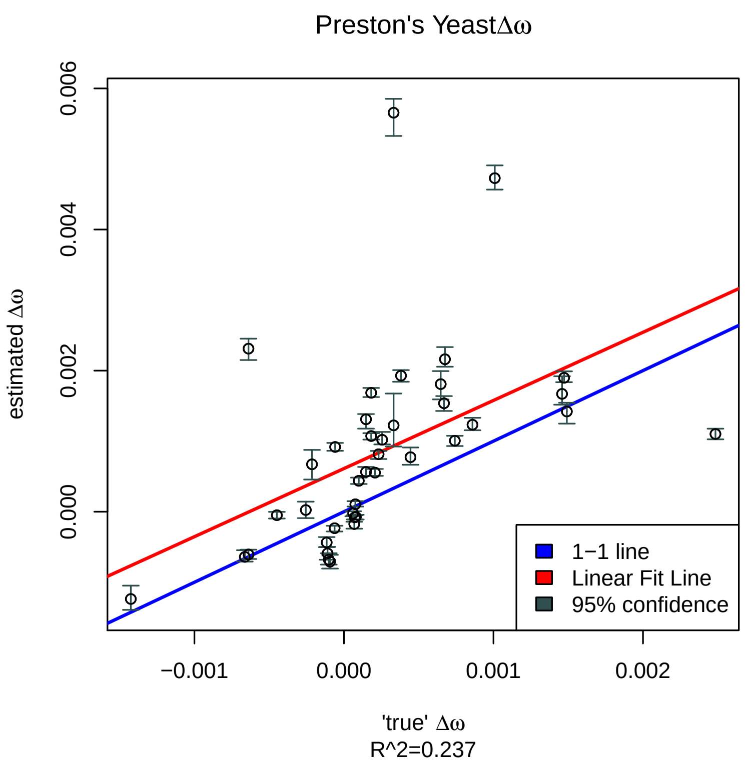


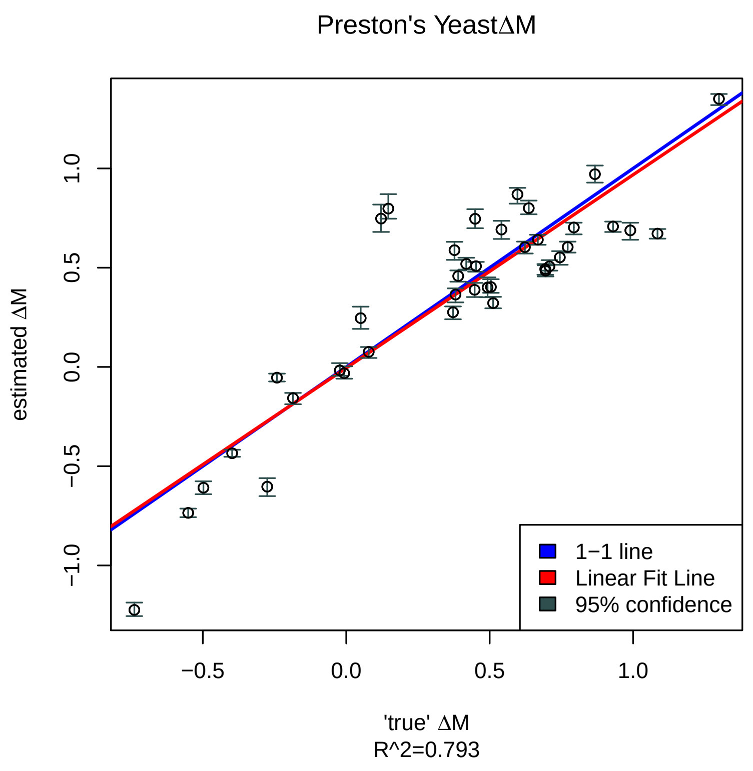
Estim. ϕ vs. Obs. ϕ
Est. ϕ (log10)
−2
−1
0
1
2
−3
−2
−1
0
1
Obs. ϕ (log10)
R^2=0.0122

Note how different the $\Delta\omega$ values are between Preston's yeast genome and the REU students' yeast genome.


REU YeastΔω
estimated Δω
'true' Δω
R^2=0.987
−0.0005
0.0005
0.0015
0.0025
0e+00
2e−04
4e−04
6e−04
🟦 1−1 line
🟥 Linear Fit Line
⬛ 95% confidence


REU YeastΔM
estimated ΔM
'true' deltaM
R^2=0.868
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
−1.0
−0.5
0.0
0.5
1.0
🟦 1−1 line
🟥 Linear Fit Line
⬛ 95% confidence

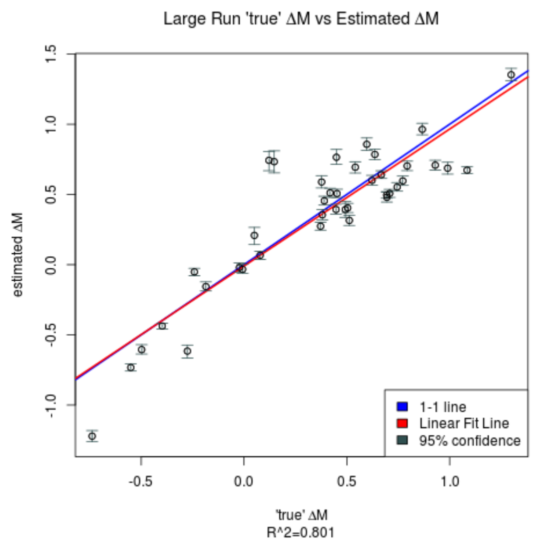

Preston's YeastΔω
estimated Δω
'true' Δω
R^2=0.237
0.000
0.002
0.004
0.006
−0.001
0.000
0.001
0.002
🟦 1−1 line
🟥 Linear Fit Line
⬛ 95% confidence

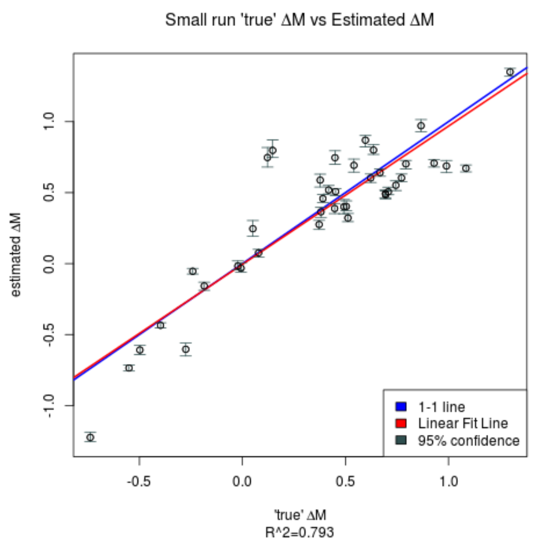

Preston's YeastΔM
estimated ΔM
'true' ΔM
R^2=0.793
−1.0
−0.5
0.0
0.5
1.0
−0.5
0.0
0.5
1.0
🟦 1−1 line
🟥 Linear Fit Line
⬛ 95% confidence

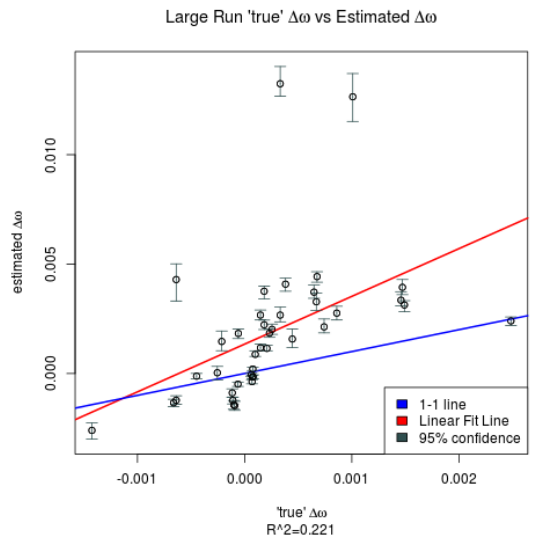
2.1.1 Longer Run?

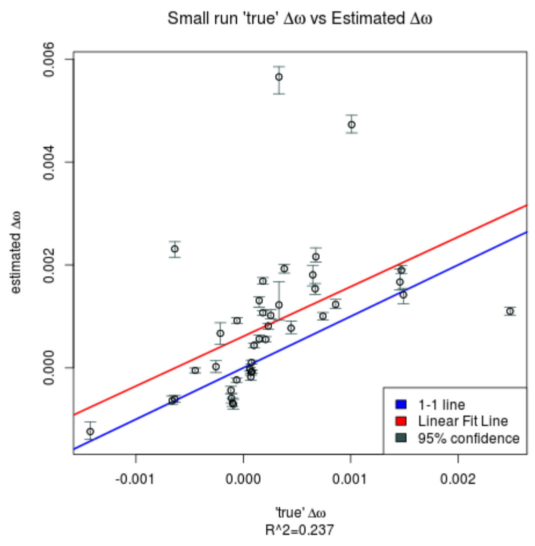
A longer run creates some interesting results. I did a run that used 7500 samples instead of 750 (which, since I'm thinning by 10, it's actually 75,000 proposals).

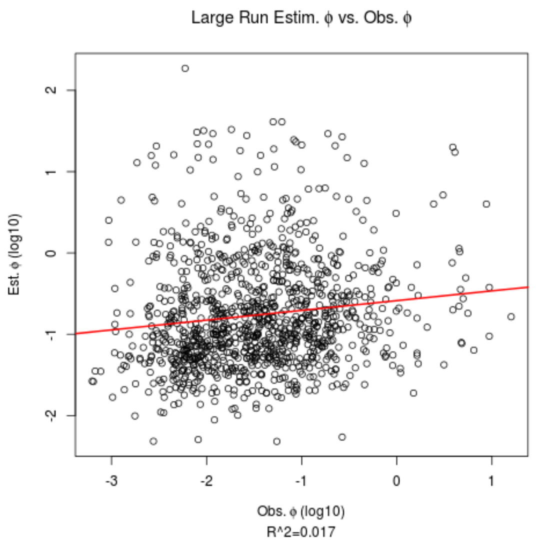
Here's the results (Large run is on the left, small run is on the right)

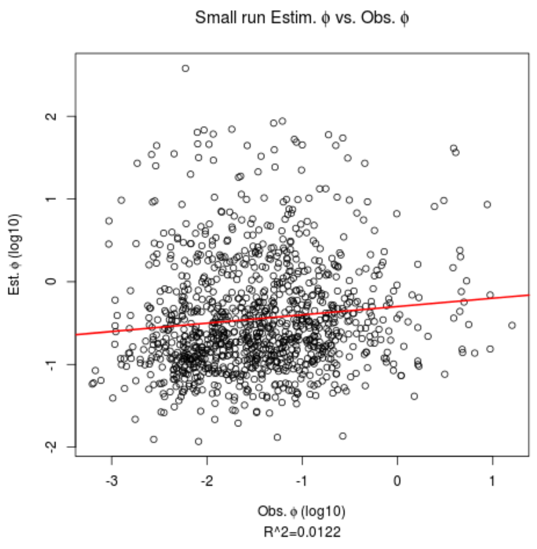


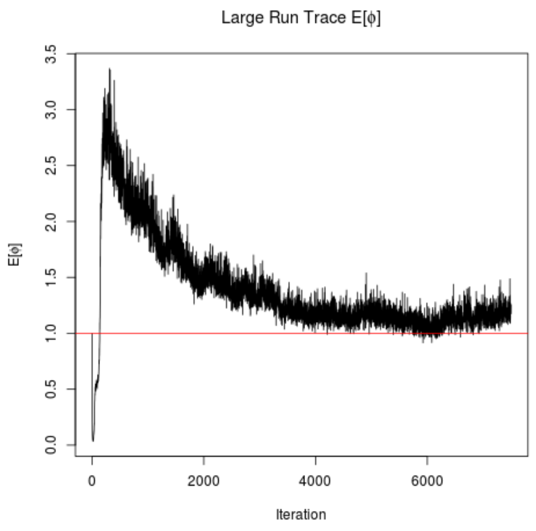


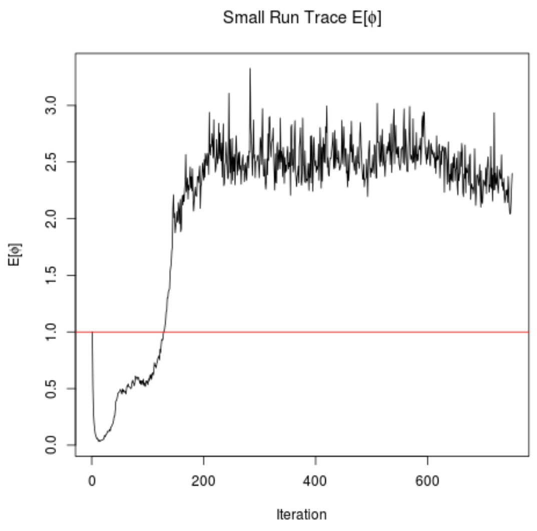












The most relevant thing in my opinion is the E(ϕ) graphs. E(ϕ) should hover around 1. in the small run, it leapt up to 3, and I was concerned this was an inherent problem in the code. But it appears that the model simply hadn't converged.

2.2 Other Genomes

2.2.1 Entire Preston's Yeast

I fixed an error in the visualize.r results by using the entire genome as the input, and letting the visualize function sort out what values belonged to who. This caused my R^2 value to jump from .02 to .5, a huge improvement.

Because of this, I'm doing a cubfits run with the entire preston's yeast genome, to see how that affecst the results.

2.2.2 Brewer's Yeast

The yeast genome used for the paper, the git repository is

`/export/home/semppr/gitrepos/wcchen/logistic_analysis.git/`

The exported files can be found in

`/home/lbrown/logistic_analysis/p01-paper`

2.3 Visualization

2.3.1 'true' values vs simulated values

Changes have been added to visualize.r, based on other plotting functions.

I've added confidence intervals (the scale of the interval can bet set in visualize.r). Cedric didn't have any functions to do so, but I was able to apply the "plotrix" package. I've also installed that package to "/home/lbrown/cubfits/Dependencies/plotrix". Everyone else should have permissions on that directory, in case someone wants to use my edited visualize.r function.

2.3.2 Fix plotCUB?

The plotCUB function right now is calculated based on the ROC model. For the estimated and observed bias, the lines do not line up with the means and confidence intervals.

CONFIRMED.

plotCUB generates the line (called predict.roc) using the prop.model.roc function. It seems to be based on my.logdmultinomCodOne.r. I've made the appropriate changes. By correctly using the $\Delta\omega$ values as $\Delta\omega$ (and not $\Delta\eta$), and also multiplying through by the mean position of the amino acid, we actually see a Codon Usage Bias!

2.4 Parallelize the Code

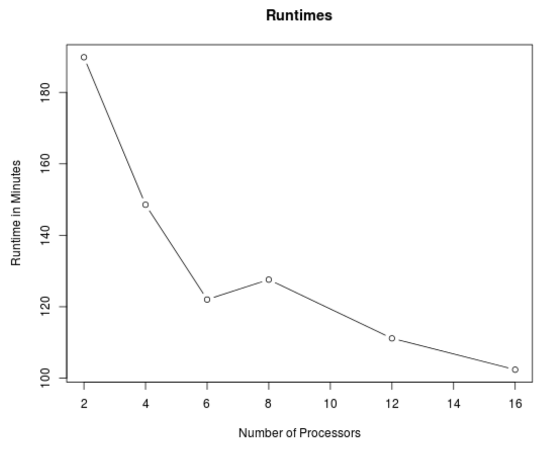
2.4.1 getOption("mc.cores")

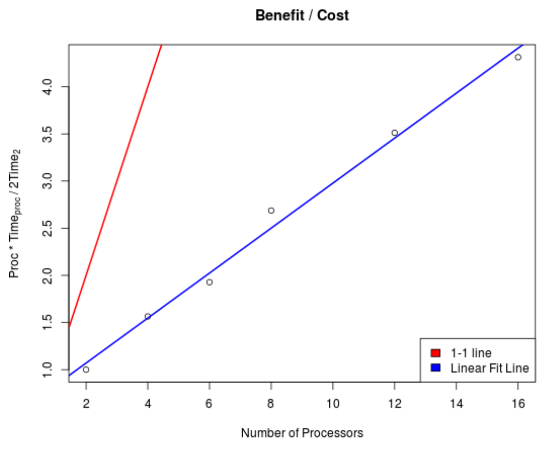
How to set the number of cores for an mclapply call? mclapply's default number of cores is getOption("mc.cores",2L);

getOption("(option)", (value)) returns the value previously set to that (option), or otherwise it returns (value). mc.cores is not set by default. So first, set option("mc.cores"=Number_of_Cores
Then mclapply should correctly get the number of cores.

2.4.2 Timing

As expected, we get diminshing returns on adding additional processors





2.5 Change probability calculation

Our results seem to indicate that the cost of a nonsense error is not high enough. The actual codon usage bias is not very high until you get to very high values of Phi. This doesn't match what we actually see.

In the code, when the posterior probability of a codon is calculated, instead of calculating

$$\Pr(c_i|\phi, i) = \frac{\exp[\mathrm{M}_i + \omega_i(a_1 - a_2)y_1 + \omega_i a_2 y_1 i]}{\sum_{u=1}^{m} \exp[\mathrm{M}_u + \omega_u(a_1 - a_2)y_1 + \omega_u a_2 y_1 i]}$$

Wei Chen calculates

$$\Pr(c_i|\phi, i) = \frac{\exp[\mathrm{M}_i - \omega_i \phi i]}{\sum_{u=1}^{m} \exp[\mathrm{M}_u - \omega_u \phi i]}$$

This was done for a number of reasons. The y_1 term is just the aggregate of the effective population, ϕ, and a scaling term $-q$. Also, the assumption was that $a_1 \approx a_2 =$ 4ATP.

2.5.1 Adding a_2

So it appears that Wei Chen never accounted for the direct cost of codon addition in the model. Maybe he was counting on it being cancelled by the scaling term q? In any case, I changed my.logdmultinomCodOne.r from

```
xm <- matrix(cbind(1, tmp.phi * reu13.df.aa$Pos), ncol = 2)
```

to

```
xm <- matrix(cbind(1, 4 * tmp.phi * reu13.df.aa$Pos), ncol = 2)
```

Since $a_2 \approx 4$.

Then xm%*% baamat should be $\mathrm{M}_i - 4\omega_i\phi i \approx \mathrm{M}_i - a_2\omega_i\phi i$ As intended.

Implementing the change seemed to have no significant effect on the model, with Preston's Yeast or the REU Yeast. The results of the former are in cubfits/misc/results/np/11-10/*4ATP.pdf, and the latter are cubfits/misc/results/ny/11-10/*4ATP.pdf

This is troubling. I'm subtracting out an additional $3\omega\phi * position$ from each mutation bias, and I'm seeing very little results in actual codon usage bias. I checked the calculations in the browser, and it's accurate.

2.5.2 $\Delta a_{1,2}$

To better account for the parameters of the model, we're going to add another parameter called $\Delta a_{1,2} = (a_1 - a_2)$, and use

$$\Pr(c_i|\phi, i) = \frac{\exp[\mathrm{M}_i - \omega_i(\Delta a_{1,2})\phi - 4\omega_i\phi i]}{\sum_{u=1}^{m} \exp[\mathrm{M}_u - \omega_u(\Delta a_{1,2})\phi - 4\omega_u\phi i]}$$

The math part of the change takes place in my.logdmultinomCodOne.r, adding an extra row to xm and multiplying $\omega\phi\Delta a_{1,2}$, which is baa[2]*tmp.phi*(new parameter)

2.5.3 How to set $\Delta a_{1,2}$

Mike suggests a small stepsize, but the big question is, when to we increase or decrease $\Delta a_{1,2}$?

I could add it to the baa matrix, and set it using the same vglm call in myfitMulitnomialOne.r that fits the M and ω values, but then it would be different for each amino acid. Of course, I could do that, and then set all of them equal to the (max/min/mean/median) value for $\Delta a_{1,2}$, but is that a bad idea? Also, this would make our VGLM call take longer, and it would disrupt our ability to fit the parameters we are already fitting. Those parameters would also be fit relative to an AA-specific $\Delta a_{1,2}$, even though $\Delta a_{1,2}$ would not be AA-specific.

2.6 Move to Newton

Newton uses R 3.0.1, all my dependencies were build under 3.1.1. Newton HAS R 3.1.0, but not 3.1.1.

2.7 Wrong Names

It's not just a problem in the chain$b.Mat data. It's in the my.fitmultinomOne.r

2.8 Wei Chen's Yeast / Real Yeast Genome

2.9 Generate my own simulated yeast, using a reverse engineered cubfits

3 Goals for next Month

1. Future Goal