Overview of Steps:
0) Goals and Bayesian Models
1) Allele Frequency calls with CI
2) Allele Frequency change
3) Diagnostic alleles and threshold tuning and predicting number of surviving clonal lines
4) Estimating clone proportions
5) Partitioning variance in clone success by metadata factors
---this is where code stops!----
6) Partitioning Allele Frequency contributions to success in various conditions
7) Concordance analysis of alleles shared by surviving clones
8) Loci associated with those alleles
9) Post-workflow sanity checks

0) Goals and Baysian Models

Target signal is: who survived and why (drift, natural selection, lab selection)?

Why all the Bayesian stats?
Baysian stats are great when you need to deal with a lot of uncertainty. We have a lot of noise. Instead of trusting our observations to be base truth, we instead use our prior expectations and knowledge of uncertainty to temper our conclusions we get from the data. That's the crux of Bayesian frameworks.

Some vocabulary helps:
prior – prior knowledge. What we expect before considering the observed data. E.g. haplotype distribution, rarity of clones, rarity of alleles, etc. This is user defined or derived from known measures like sequencing error rate, mapping error rate, call likelihood, knowledge of genome, etc.

likelihood – a measure of how reliable our observations are. How likely are we to observe something if our prior is true (written p(O | H) or probability of observation given hypothesis). E.g. how likely are we to observe a diagnostic allele if the clone is present (~15% probability due to low coverage) – p(allele at this loci | clone is present) = 0.15.

Posterior – updated knowledge. What we expect given our prior understanding and the new information from observation. E.g. Probability of observing a clone with 10 diagnostic loci given the expectations of clone rarity, missed calls, etc.

1) Allele Frequency calls with CI

What we did originally:
Originally, we used the simple calculation:

Allele Frequency = AO/(AO + RO)

where AO is alternate allele calls and RO is reference allele calls.

Example using numbers (from our data):

AO=642	Alternate allele observations (number of alt bases observed)
RO=510	Reference allele observations
DP=1152	Total read depth (AO + RO)
AC=34	Estimated total alternate alleles (not calls, actual count from model)
AF=0.53125	Estimated allele frequency (probably equal to AC / AN)

AN=64 Total number of alleles (2 × individuals if diploid)
NS=32 Number of samples (individuals or pools)

Our original calculation, ignoring uncertainty.

AF = AO/(AO+RO)= 642/1152 = 0.557

However, this formula assumes that:
* all reads are error free
* there is no sampling noise
* no alleles are missing or biased by mapping
* read count is bullet proof

Obviously, this can be improved. What the Feder Lab did is move to the Bayesian framework: instead of calculating the single point measure of an allele at a loci, we seek to determine the *distribution of possible allele frequencies that would give us the observed counts*. This moves us away from a point estimate and toward estimating posterior distributions over allele frequency values (just means that we could plot probability on y and allele frequency on x to get a distribution).

What Feder Lab did:
Taking a cue from https://onlinelibrary.wiley.com/doi/full/10.1111/ele.12460, we want to adjust how we call allele frequency to build in some robustness to uncertainty due to low numbers of observations and uneven sampling of loci.

In that paper, they used:

$$p = \sum_{i=1}^{n} \sum_{j=1}^{3} g_j \times p(g_j|D_i),$$

gj is number of alternate alleles, constrained to 0,1,2 for homozygous ref, het, homozygous alt
p(gj | D) is the probability of observing that allele count given the data and a binomial model

Effectively, they are weighting the probability of the counts observed by the probability of those counts given the data, then summing that probability across all genotype options (0,1,2) and individuals (n).

For example:
If the probability of RR is 0.1, RA is 0.6, and AA is 0.4 (Reference/Alternate) given the reads observed, the calculation for that individual is:

alternate alleles expected = (0*0.1) + (1*0.6) + (2 * 0.3) = 1.2
reference alleles expected = (2*0.1) + (1 * 0.6) + (0*0.3) = 0.8

sum across individuals = (1.2 + 0.8 + 1.0)/(2*3) = 3/3 individuals = 1.0 average

That is summed across all individuals and divided by 2 to account for diploidy. So, their predicted alternate allele frequency in this made up situation would be 0.5.

The p(gj|Di) can be determined from many things, including base call scores, mapping quality, error models, etc. GATK does some of this within HaplotypeCaller, providing PL (phred-like score), GP (genotype probability), and GQ (genotype quality) scores in the vcf.

Feder's Lab estimated individual genotype likelihoods using GATK with a binomial distribution and a uniform prior. We can skip the individual genotype call likelihoods (hard to do in pools) and work with allele counts while still keeping the uniform prior and a beta-binomial distribution to help guard against uncertainty we know is in this data.

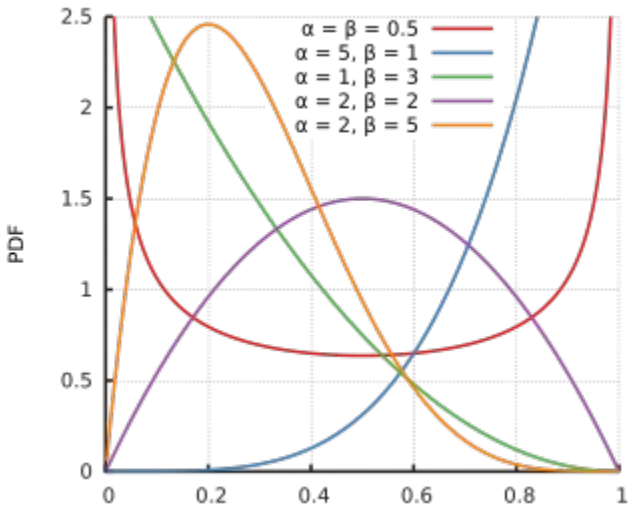
Modifying our approach:
We did not use GATK (because I hate it and freebayes is much less frustrating), we don't have GP scores. However, we do have direct measures of number of AO and RO for each sample, so we can actually just skip that computation and use our direct numbers.

The probability of seeing an alternate read depends on the number of reads and the probability of drawing an alternate. We do not want to treat the observed AO as base truth, but the slightly inaccurate result from the random process of subsampling n reads from the true distribution. We're not estimating what AO is based on what we see, but using our observation to estimate what the true value was given the *uncertainty in our sampling* and our prior knowledge of subsampling.

Assumptions we can make about our data to help set this up:
* Each read is assumed to be an independent draw.
* The number of reads is the sum of AO and RO (or just DP)
* Observations are either AO or RO, no other option

The probability of AO observed can therefore be based on a binomial distribution. Binomial distributions are from independent, yes-no observations. More specifically, we can use a beta binomial distribution.

Use a Beta prior with our previous calculation in a Bayesian framework
Beta distributions are commonly used for counts, and this serves as our *prior* expectation – what we expect before doing anything. We have prior knowledge that subsampling allele frequency looks like a binomial distribution, which we can define with a beta prior to tune the expected shape of that binomial. We assume when we subsample from our true allele frequency, the distribution of possible results looks like a bell curve defined by beta(a, b).



Beta distributions have two terms (a, b) which allow us to adjust the curve based on some additional expectations:

* Beta (2,2) – alt and ref are equally likely
* Beta(2, 5) – alt is much more rare than the reference
* Beta(0.5,0.5) – called Jeffrey's prior which allows for even more uncertainty handling?
* Beta(1,1) – favor neither side, called **a uniform prior**

The last (uniform) is the common one to use, so I'm defaulting to that. It is also what Feder uses. This means if we were to subsample from a true distribution of, say 0.5 AF as in the figure, we would see a ~normal distribution of observations with increasing probability as we approach the maximum probability value at our true AF.

Our updated formula:
Through more Bayesian math, we can estimate the "p hat", or sample proportion, which in our case is the proportion of the reads that are truly AO. That's great, because "p hat" ends up being our AF!

We can assume our true AF (p) is drawn from a Beta binomial distribution, as we discussed above.

p ~ Beta(a, b)

our beta posterior calculation becomes:

P(p | AO, RO) ~ Beta(a + AO, b+RO)

the mean posterior (least means estimator) gives us p hat:

$$\hat{p} = \frac{\alpha + \mathrm{AO}}{\alpha + \beta + \mathrm{AO} + \mathrm{RO}}$$

This is now a **Bayesian corrected estimate of allele frequency**, where a and b are the alpha and beta constants from the Beta distribution (user defined). In our example from above then becomes:

p = (1+642) / (1+1+642+510) = 0.557

This doesn't make a ton of difference in the example, but the slight tweek helps guard against weirdness in loci with smaller sample size (stabilizes mean-variance relationship, which is important).

Imagine we have 3 AO counts and 1 RO counts. 3/(3+1) = 0.75 using our original calculation. However, with the above, we get:

1+3 / (1+1+3+1) = 0.5

This change accounts for coverage variance, reduces skew in low counts, allows you to input prior information (is the alt rare, etc). If you think about this with less Bayesian nonsense, we are basically averaging our expectation (uniform) with the observed frequency, with a weaker reliance on the prior if we have lots of observed data, but a higher weight on it if we have limited observations. Basically, we are tempering how much we update our prior knowledge (uniform distribution) based the level of uncertainty in our counts. If we wanted to make the prior have stronger influence, we can just increase it to beta(2,2), which keeps it even, but makes it even less flexible in light of new evidence.

More importantly, gives us a **full distribution** over the allele frequency – which means we can compute CONFIDENCE INTERVALS!

We have a prior expectation of the count distribution we should see if we were to sample the true value, and what we observed. We can use that to compute the probability of observing AO and RO given the distribution over a range of possible 'true' AF values. In other words, **we can determine what range of AF would give us our AO/RO measures 95% of the time**. That's a confidence interval!
Think about the above case where you have 3 AO and 1 RO. That observation could come from a much larger range of distributions, meaning it will have a much wider confidence interval to capture 95% of the likelihood. The real example with 642 AO and 510 RO will have a narrower set of distributions that could likely result in that sampling. I calculate these with code, as I don't want to have to calculate the ranges by hand so I don't know the math here, but if you run the real example through, you get:

AF = 0.5571
CI = 0.5293 – 0.5845

Interestingly, this CI covers our original estimate. Also, it's pretty narrow, which makes sense given how many observations we had at this loci (1152).

NOTE: There is one other way to calculate this since we used FreeBayes. You could in theory use AC/AN, where AC is the reported *estimated total alternate* alleles by freebayes and AN is the total number of alleles (2*diploid value submitted). This can be a bit dependent on the number of diploid genomes we expect (clone number) which we know is a bit tricky. Just for completeness, the calculation is

AF = AC/AN = 34/64 = 0.53125 (AF)

freebayes reports this as AF. This is also within our above confidence interval, but I feel less good about this in general since we have unknown numbers of genomes in the buckets at this point.

Function: extract_AF(vcf_data)
Input: filtered vcf file, e.g. All_buckets_filtered.vcf.gz
Output: AF matrix

General Instruction: After you have created a merged, filtered vcf with all of your samples, run this code to import that data into R and calculate an improved measure of AF using a beta binomial prior.

2) Allele Frequency change
Since we have averages with intervals, we can compute differences with some level of confidence as well. We want to use non-parametric stats here, as we can't really assume normality. We also want to make sure we include the uncertainty in the analysis and we want to avoid 'fragile p-values' due to small sample sizes. We can do this with a Monte Carlo Simulation approach, much like the Feder paper.

In this case, we sample the two distributions thousands of times, calculating the deltaAF for each pair of samples. We now have a new distribution for deltaAF with its own mean, CI, and probability.

The result is an intuitive "there is a X% chance that the allele frequency is higher in T2 than T4". You can then plot this, look for large differences that do not have CIs that cross 0 (if 0 is in the CI, then you cannot rule out 0 change).

Is this the final answer then? Could be! But any change we see is going to be a reflection of shared alleles in successful clones, not necessarily the loci that are most driven by the environmental condition. This is the result of having no recombination in the clonal populations. So while interesting and definitely useful, I think this will still be kind of messy with the essentially genome-length linkage groups.

Function: monteCarloDeltaAF(AFmatrix,
metadata,
sample_col=”Sample”,
compare_col=”TimePoint”,
group1 = 1,
group2 = 4,
n_iter = 10000,
ci_level = 0.95)

Input: AF matrix from above
Output: results$mean_deltaAF, results$CI, results$p_values

Function: plotSignificanceDensity(results_from_monteCarlo,
window_size=100,
title="Proportion of Significant Loci per Window")

Input: Results from monteCarloDeltaAF(…)
Output: Proportion of significant loci (p<0.05) per n-locus window along linear order

General Instructions: After getting an AF matrix, calculate the change in allele frequency between time points to identify allele frequency change in each paired sampling (T2 v T4). Plot if you would like with plotting function :)

3) Diagnostic alleles and threshold tuning and predicting number of surviving clonal lines

This part you know and did already – identifying how many clones are in each sample is very useful for the next step (estimating proportions). You identified private alleles for each clone, and then used that to identify likely clones surviving. You are working on tuning the threshold of how many diagnostic loci must be observed to call it a present. I’m going to think about this more, but I think this is perfectly fine at the ~15% you were using.

However, I could not let good enough alone, so I did some estimating of probability of missing clones given different number of loci.

Baysian estimation of loci needed to confidently call clones:

Givens:
Bayes Theorem – just using the standard probability for this one.

Prior probability a clone is in the pool, taken from your manual estimates in T2 and T4:
P(C) = 0.8 (T2)
P(C) = 0.08 (T4)

Prior probability that a diagnostic locus is observed when the clone is present, taken from 200 genotypes with 30x coverage:
P(L | C) = 0.15

P(-L | C) = 0.85 → probability of not observing a locus when clone is present

Prior probability that a locus is observed when the clone is not present, driven by the sequencing error rate (false positive)

P(L | -C) = 0.0001

P(-L | -C) = 0.9999 → probability of not observing a locus when clone is not present

Probability of a clone being present if a locus is detected:

$$P(C|L) = \frac{P(L|C) * P(C)}{P(L|C) * P(C) + P(L|\neg C) * P(\neg C)}$$

P(C|L) = (0.15 * 0.8) / (0.15*0.8 + 0.0001 * 0.2) = 0.99983

If you detect a diagnostic locus, there is a 99.98% probability that the clone is present. Also note, these values don't change significantly if we assume P(C) = 0.08 as we see in later buckets.

Probability of MISSING a clone by evaluating one locus:

$$P(C \mid \neg L) = \frac{P(\neg L \mid C) * P(C)}{P(L|C) * P(C) + P(\neg L|\neg C) * P(\neg C)}$$

P(C | -L) = (0.85 * 0.8) / (0.85*0.8+0.9999*0.2) = 0.7728

This is obviously too high, so we want to see how many loci we need to hit a threshold of reasonable probability.

How many loci do we need to miss to get have confidence that it is missing:

$$P(C \mid \text{all } \neg L) = \frac{(0.85)^n \cdot 0.8}{(0.85)^n \cdot 0.8 + (0.9999)^n \cdot 0.2}$$

where n is the number of failed loci, and all -L means that all n loci failed to detect the clone. This represents how probable it is to miss a clone if all n loci are not detected. We want to find a solution for n where P(C|all -L) is below 0.1 and 0.05.

Solving for various cases:

T2 (80+ clones likely surviving), P<0.1 = 11 missed loci is sufficient to detect a missing clone
T2 (80+ clones likely surviving), P<0.05 = 42 missed loci is sufficient to detect a missing clone

T4 (~8 clones likely surviving), P<0.1 = 4 missed loci is sufficient
T4 (~8 clones likely surviving), P<0.05 = 7 missed loci is suffcient

How many loci do we need to detect to have confidence that the clone is present:

$$P(C \mid nL) = \frac{(0.15)^n \cdot 0.08}{(0.15)^n \cdot 0.08 + (0.0001)^n \cdot 0.92}$$

Solving for various cases:
T2 (80+ clones), P<0.1 = 1
T2 (80+ clones), P<0.05 = 1

T4 (~8 clones), P<0.1 = 1
T4 (~8 clones), P<0.05 = 1

In short:
Detection is easy with diagnostic SNPs, but missing clones is a bigger issue. You can detect clones with ~1 positive result. But you need 11-42 SNPs missed to truly call it missing. For our purposes, we're more tolerant of false positives (more clones predicted than are really there), so 11 missed to call it missing is totally reasonable. These numbers will act as a sanity check to the next step.

4) Estimating clone proportions
Useful information for determining proportions:
* we have 100 clones with a vector of biallelic/filtered SNP genotypes to act as a profile for each clone
* we have bulk SNP data of the pooled data, which are additive mixes of the clone profiles
* we have an estimate of the predicted clones surviving in each bucket

We can use non-negative matrix factorization or Bayesian regression to deconvolute this data into a linear combination of genotypes weighted by their frequencies in the mixture. Essentially:

observed bulk AF = Sum across all clones (clone proportion * clone SNP profile)

I'm going full Bayes on this, so I set up a Bayesian regression function in R to do this. We could do NNMF if desired. Bayes gives us a probabilitstic estimate of clone proportions with uncertainty, and won't be bothered by missing clones in some buckets (which we know happens).

We want to model the observed AF in bulk as weighted mixture of known clone profiles, where the weights are unknown and drawn from some distribution (uncertainty of evenness is handled!). This is handled in the code provided. After we fit the model, we can extract the posterior, giving us clone proportions with 95% CI intervals and p values. This also is perfectly happy with 0s for clones that fall out of the mixture (with uncertainty).

I tested this with some simulated data (which you cal also make with the function below), and am running it on real data.

Function: makeDeconTestData()
Input: N/A
Output: results$G, results$bulk_mat

Function: loop_deconvolute_clones(clone_matrix,

bulk_matrix,
n_iter = 2000,
chains = 4,
cores=4)

Input: AF matrix from individual clones, list of AF values for target mixture
Output: results$estimates, results$ci_lower, results$ci_upper, results$pvals
prints number of detected clones to screen to sanity test

Function: deconvolute_clones(clone_matrix,
bulk_matrix,
n_iter = 2000,
chains = 4,
cores=4)

#takes ONLY one input vector to deconvolute. It will run on multiple, but it will output average results, not one per as above. This is the function the above loop calls. You likely won't need it.
Input: AF matrix from individual clones, list of AF values for target mixture
Output: results$estimates, results$ci_lower, results$ci_upper, results$pvals
prints number of detected clones to screen to sanity test

General Instructions: Now that you have a corrected measure of deltaAF that accounts for uncertainty, you can deconvolute the observed signal AF into individual clone proportions (since observed AF = clone proportion * clone AF). The goal of this step is to end up with a matrix of clones (rows) x buckets(columns), with entries being the proportion of that clone in that bucket.

####################Stopped here with code and testing#################
5) Partitioning variance in clone success by metadata factors
Short version – we want to put the matrix we just created in step 4 into variancePartition. We split the signal into different clones, now we want to split it into different factors – Time, Replicate, and Treatment (from metadata table).

6) Partitioning Allele Frequency contributions to success in various conditions
Short version – the AF driven by, say, salt is equal to the sum of each clone's contribution. Clone contribution is equal to clone proportion * salt proporiton * observed AF. This will allow us to separate out the signal for Salt, Temp, Salt/Temp (all Treatments), bucket-level variation (?), and Time (drift?).

Until testing, this is just an outline.
7) Concordance analysis of alleles shared by surviving clones?
8) Loci associated with those alleles
9) Post-workflow sanity checks