Assignment 1

The goals of these exercises are the following:

1. Get you to use Rmarkdown or Jupyter notebooks to write reports
2. Get a sense of your coding & statistical background and
3. Give you some intuition about some key probability distributions involved in genomics.

The reports should include the code and your comments/observations about your approach and results.

Lander-Waterman Theory & Genome coverage

Genomic mapping by fingerprinting random clones: A mathematical analysis

In designing a sequencing experiment we:

1. Know the size of the genome G (assume haploid)
2. Can decide how many DNA fragments to measure N
3. Can control the length of the fragment L

If we assume that the reads are drawn uniformly at random from a genome[1] then the probability that the random position is covered by a read is $p = L/G$ (if we ignore boundary condition close to telo-/centro-meres since $G \gg L$). We can thus use the "coin" model we developed previously (aka the Binomial distribution) to answer the following questions:

1. What is the expected depth of coverage of a random position? In other words, how many reads to we expect to overlap a random position?
2. Draw the probability density for this coverage for $L = 600$ (~ Sanger Sequencing), $G = 3 \times 10^9$ (~ human DNA) and N=10 million reads. Using the expected coverage (`mean`) for these values, draw (in the same figure) the Poisson distribution that best approximates the distribution you calculated. Comment on the quality of approximation.
3. Using the Poisson approximation, estimate the mean depth required to cover the genome by 99% (or probability of gap = 1%). How many reads of $L = 600$ does this coverage require?

Negative Binomial Distribution

In the coin model we used so far we fixed the probability of heads (`theta`) and the number of trials (`n`). But we can decide to conduct the experiment differently, we can fix the number of heads (1) and count how many trials we had to perform to get them.

1. Write a code that simulates such a process. The code should take as input 2 parameters: the probability of heads (1) `prob` and the require number of successes `size`. It should return the number of trials required to get there minus the `size` (we want the possible count to include 0).
2. Run your code multiple times to get a histogram of the distribution. Plot the density of negative binomial (using the same parameters) on top of the histogram.
3. Try to do inference on p using your own model, ie use your model with a known `prob` (p) to generate random sample and then estimate the value of $\hat{p}$. Repeat the estimate many times to get a histogram of your sampling distribution. Comment on the bias of your estimator.
4. *Optional*: Estimate the p using `MASS::glm.nb(x ~ 1)`[2] and compare it to your estimate. *Note*: `MASS::glm.nb` coefficient is the log of the expected number of trials ($E[\log(n)]$).

[1] they are not, but imagine we focus on the "good" part of the genome

[2] if you are using Python you need `statsmodels.discrete.discrete_model.NegativeBinomial.fit`

Feature Counts & Library Complexity

Imagine a set of C (for complexity) distinct DNA fragments (aka library) each of which has n_i copies such that $\sum_{i=1}^{C} n_i = N$. A sequencing experiment consists of picking m fragments at random (sample) and "reading" them. For ease of computation we are going to assume that N is much larger than m[3] so that the pool is un-depletable. Thus each fragment is characterized by its relative abundance $p_i = \frac{n_i}{N}$.

1. Write a function that generate random samples from this model. The function must take as input a vector `prob` (the size of the vector is C) of relative abundances and a sequencing depth `size` and return a vector of counts for each fragment (including 0 counts).

You can generate a discrete probability distribution $P = (p_1, \ldots, p_n)$ by generating N positive random numbers n_i and then setting $p_i = n_i / \sum_i n_i$.

2. For a given C and `size` (eg C = 20k protein coding genes, `size`=1M reads):
    i. Generate 1 random sample of C read counts from a "null" distribution where all the fragments have the same relative abundance $p_i = 1/C$.
    ii. Plot a histogram of the sampled counts and use their mean to draw a Poisson density function on top of it. Comment on the quality of the approximation.
    iii. Repeat the previous steps but this time use a gamma distribution to generate random abundances. Compare the resulting histogram to the best Poisson approximation. *Suggestion*: Choose parameter so that gamma has a "nice" bell shape.
    iv. *Optional*: try to fit a negative binomial distribution to the counts. Comment on the difference between the 2.
3. We say a fragment is detected if the corresponding read count is greater than 0.
    i. For a fixed complexity C and abundance vector `prob` test different number of sequencing depth `size` and estimate what percentage of C is detected each time. Plot this relationship.
    ii. Repeat this processes for the 2 sets of relative abundances (gamma and fixed) and compare them.

In a real experiment we do not know how many 0 are in our count vector so we have to estimate it from the rest. If we have an estimate of $P(X = 0)$ then the expected number of observed reads is $C_{\text{obs}} = (1 - P(0))C$ for which we can solve for C and get an (probably conservative) estimate of library complexity. Estimating $P(0)$ is done via a *truncated* negative binomial regression. In R that could be done via the VGAM package, in Python I do not know, in practice via Picard's EstimateLibraryComplexity command (which estimates C analytically).

[3] We can always duplicate the fragments with PCR and increase N as much as we like. We have to assume though that PCR amplifies all C equally (spoiler alert, it does not).