week 07: differential gene expression analysis

our goals this week

- An introduction to the analysis problems that come up when we try to decide whether a gene is differentially up- or down-regulated in some condition, using RNA-seq data: **differential gene expression analysis** (DGEA).
- Data analysis beyond Python: we're going to install R, Bioconductor, and the edgeR package for differential gene expression analysis. We'll still use Python, but we'll use Python to export and run R scripts, as a low-overhead way to use R packages in our analyses.

the problem

We want to detect when a gene is differentially expressed in one condition versus another. To do that, we need to understand how variable that gene's measured expression level is to begin with, in each condition.

That is, suppose we just did two RNA-seq experiments, one for wild-type and one for a mutant. For our favorite gene *calico*, we see 50 mapped reads in wild type, and 100 mapped reads in mutant. Can we say that *calico* is two-fold overexpressed in the mutant? No, because for all we know, if we measured *calico* in different samples of wild-type, we might get 25, 50, 100, 70... we might see that 100 is entirely normal, within the observed range of variation for wild-

type samples.

We need to understand the variation we expect across different samples *within* each condition, before we can decide that a gene's expression is significantly different *across* conditions.

To understand variation, we need *replicates*. We need to take multiple measurements of each gene in each condition.

People distinguish two kinds of replicates. *Technical* replication means measuring the same biological RNA sample more than once. *Biological* replication means doing the whole RNA-seq experiment over again on another sample of the same cells. You might study technical replicates if you were interested in developing experimental protocols that minimize noise. For differential gene expression analysis, we need biological replicates. We're trying to decide whether any genes in the mutant sample are behaving differently than we'd expect, statistically, if we just did another experiment in wild type.

Besides the condition you're trying to test, you're battling against many other factors can make some gene expression levels go up and down. These include things like growth conditions (food, water, temperature, light), genotype (comparisons of outbred animals show more variation than inbred), noise and vibration (animals get stressed), how long your samples sit on your bench, variation in reagent lots, and so on. RNA-seq experimental designs need to either assure that these variables are held constant for your control and test samples, or make sure that they're equivalently sampled. You don't want to do your controls on Tuesday and your test samples on Wednesday. When you get differences because your samples differed by some factor other than what you're trying to test, that's called *batch effect*. Batch effect is a big problem in RNA-seq experiments.

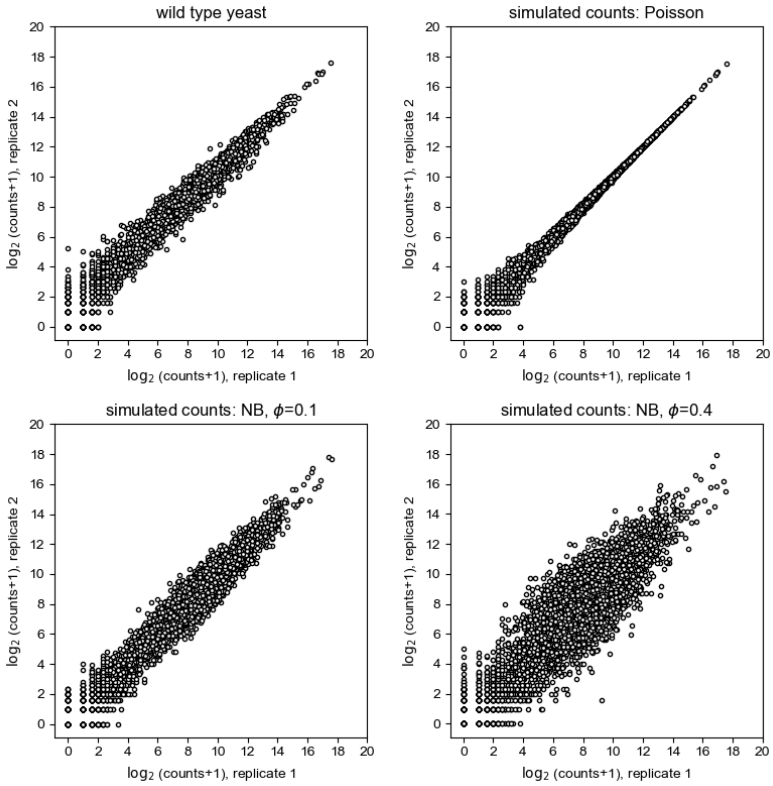
Many biological replicates are required to measure variance for each gene -- tens, at least. Almost nobody does tens of replicates. Usually, people only do two or three. How can we estimate the expected variance for each gene's expression level from just two or three observations? We can't, not without making some pretty strong assumptions. This is what **differential gene expression analysis (DGEA)** methods do.

what some real biological replicates look like

There are only a few examples where someone has done enough biological replicates to really measure normal variation for each gene. One such data set comes from Schurch *et al.* (2016), who did 48 biological replicates of wild-type yeast *Saccharomyces cerevisiae*, and 48 biological replicates of a *snf2* mutant. The *snf2* mutation causes massive changes in transcriptional regulation of thousands of yeast genes.

You can download the Schurch count data for each of 7126 yeast genes in the wild-type condition here as a tab-delimited file. It has 49 columns: the first column is a code for the gene name, and the remaining 48 columns are wild-type biological replicates.

In the figure below, I've plotted their replicate 1 versus 2 (top left). I've also plotted what you'd get if variation were simply Poisson distributed (i.e. if the only variation came from finite count sampling noise), and if it were overdispersed (negative binomial, with two different dispersions $\phi = 0.1$ and 0.4.



These are very well-behaved data, as these things go. Many RNA-seq biological replicates look worse than this. Schurch *et al.* comment that all their Pearson correlation coefficients (R) are > 0.97 for all pairwise comparisons, whereas other published experiments can show $R \sim 0.6 - 0.9$. The user's manual for edgeR comments that typical values for the dispersion ϕ are 10^{-4} for technical replicates, 0.01 for genetically identical organisms (like yeast) and 0.16 for outbred organisms (like humans). (These suggestions seem a little optimistic, in my experience.)

The axes of these graphs are on a log2 scale. You can see that observed counts can easily vary by 2-4x across replicates even for this data set, and can vary by 8-16x across replicates for higher dispersions. (How the heck do people ever draw conclusions from 1.2x-fold over- or under-expression, from two or three replicates? Who knows.)

definition of the inference problem

We observe **counts** y_{gi} for genes g in samples i.

In an RNA-seq experiment, the counts are the number of reads that mapped to gene (or transcript) g. There are other experiments that generate digital tag counts for transcripts or genomic regions too, and the same ideas will apply to differential analysis of such count data.

The samples are from two **conditions**. You might represent the two conditions by something like 0 or 1 in the math and the code, but biologically they would be something like wild-type versus mutant, or control versus treated. (There might be more than two experimental conditions -- a time course, for example -- but let's stick with the two-condition case for now.) Each of our samples is associated with a condition: c_i is 0 or 1. This is a simple version of what's called the *design* of the data table.

There'd typically be on the order of 10-20K genes, but typically only a small number of samples (typically on the order of 2 or 3 biological replicates per condition).

We want to ask the question of whether a particular gene g is *differentially expressed* in one condition versus the other.

We imagine that there's a generative model θ that generates the observed counts. If the gene isn't differentially expressed, all samples use the same model θ. Call this hypothesis H_0: it's our **null hypothesis**. If the gene *is* differentially expressed, then there are two condition-specific models θ_0 and θ_1. Call that hypothesis H_1; that's the hypothesis we want to test.

Bayes tells us:

$$P(H_1 \mid \text{data}) = \frac{P(\text{data} \mid H_1)P(H_1)}{\sum_h P(\text{data} \mid H_h)P(H_h)}$$

So, we're going to start thinking about this in terms of the **likelihoods** of the two competing

models:

$$P(Y_g \mid H_1) = \prod_i P(y_{gi} \mid \theta_{c_i})$$

$$P(Y_g \mid H_0) = \prod_i P(y_{gi} \mid \theta)$$

And so, we'll want to start by assuming some fundamental form for a distribution $P(y \mid \theta)$ that generates observed count data, given parameters θ, so we can calculate these likelihoods.

Poisson distribution probably isn't good enough

We started discussing this in week 05 when we introduced the negative binomial distribution for the first time. Let's pick up that discussion again, and elaborate on it.

If we could think about generating counts like pulling balls out of urns, then the differences we see across replicates of the same condition would just be **technical variation** due to finite count sampling error. We would expect the observed counts y_{gi} to be Poisson-distributed around a mean μ_g. Early DGEA analyses often assumed Poisson statistics.

The Poisson distribution only has one parameter. The variance σ^2 of the Poisson is equal to its mean μ. This is a strong assumption about σ^2.

You can see in the plot of the Schurch data above that real data aren't Poisson distributed. In real biological data y_{gi} from biological replicates have more variance than the Poisson predicts: they are **overdispersed**. This shouldn't surprise you. Again, if the samples are outbred organisms with different genotypes, for example, we get expression variation due to genotype variation. We also get variation from a host of other

environmental and experimental factors that are changing when we do biological replicates.

Just as a practical matter, we want a distribution form that allows us to control the variance separately from the mean, so we can model this overdispersion. Empirically, two different distributions have been seen to fit the biological data reasonably well: the **negative binomial** distribution and the **lognormal** distribution.

Table 1 in [Schurch et al, 2016] summarizes the statistical assumptions of many leading DGEA packages (as of 2016). We're going to concentrate semi-arbitrarily on one of those packages, *edgeR*. *edgeR* assumes that counts follow a negative binomial distribution.

(You can look back to week 05 for more about the mathematical definition of the negative binomial distribution in terms of its two parameters: mean μ, and dispersion ϕ.)

mean counts depend on library size

How many counts we see for gene g in sample i depends on how many total reads we measure in sample i, of course; if we sequence twice as many reads total, we'll see twice as many counts g. If we based our "differential expression analysis" literally on just comparing y_{gi} across samples, we could see statistically significant differences for the trivial reason that we have different library sizes.

So let's think about this a little more.

The number of counts we expect for gene g in sample i, μ_{gi}, is the *total number of counts* we collected in sample i times the *proportion* of the total number that map to gene g. So let

$$\mu_g = N_i \nu_g$$

where N_i is the unknown library size for sample i, and ν_g is the proportion of reads that map to gene g.

What we're really interested in is whether ν_g differs in different conditions, not μ_g. The N_i is pesky; we don't really observe it directly. It's definitely not the total number of reads we have (unmapped + mapped), because every RNA sample differs in how much random crud it has in it (bad reads, reads from other organisms) and we're certainly not interested in differences in the proportion ν_g that are due to that. Intuitively, it seems that it's more correct to imagine that N_i is the total number of *mapped* reads.

This doesn't seem to be exactly what *edgeR* does, nor other DGEA packages, but it's close. There's a step that they call *normalization* that seeks to correct the observed y_{gi} for unequal library sizes, to make it look as if they all came from the same total library size. (I won't go into details of their calculation, which is partially described in the edgeR user guide.)

edgeR does this normalization step automatically; you don't call any *edgeR* function to do it.

relative abundances depend on other genes

Remember that RNA-seq measures relative abundances like ν_g in a sample, not absolute RNA abundances per cell. If you overexpress even just one gene, the relative abundance of *every other* gene has to go down. Those changes could be large enough to be "statistically significant". We aren't interested in measuring statistically significant "indirect" effects on relative RNA abundance; we just want the "direct" effects.

By default, *edgeR* does not deal with this, but it has an optional second normalization step that it calls *trimmed mean of M-values* (TMM) normalization.

Again we won't go into the calculation. In essence, *edgeR* assumes that most genes are *not* differentially expressed across samples, and it seeks an *effective library size* by seeking to minimize the number of genes that do show expression differences.

A TMM normalization step is optional in *edgeR*. It is done by calling the `calcNormFactors()` method.

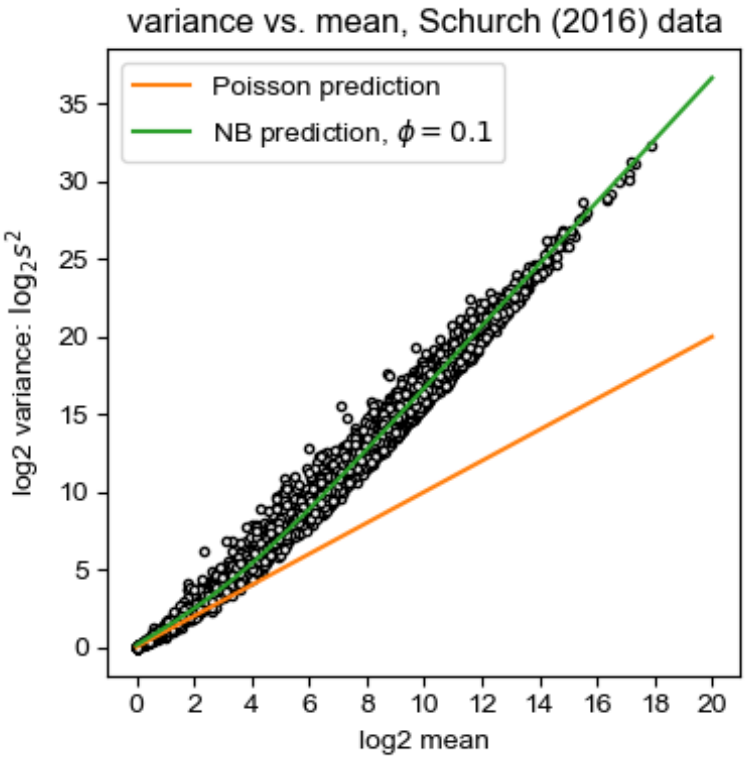
common dispersion assumption

In order to get away with a small number of biological replicates, we make a huge assumption: that there is only one **common dispersion** ϕ for all genes, rather than gene-specific overdispersion terms.

Is that a reasonable assumption? The NB says that if there's a common dispersion, we expect a particular **mean-variance relationship** in RNA-seq count data: as the mean counts μ go up, the variance should go up quadratically:

$$\sigma^2 = \mu + \phi\mu^2.$$

Empirically, this does indeed fit well to observed RNA-seq count data. In the figure below, for each yeast gene in the Schurch (2016) wild-type data, I've plotted the sample mean and the sample variance (s^2) in the 48 biological replicates. Each of the 7126 points is one gene:



For Poisson-distributed count data, we expect the variance to be equal to the mean, $\mu = \sigma^2$. That's the orange line. The real data are clearly overdispersed relative to the Poisson. The green line shows the negative binomial fit for $\phi = 0.1$, and it's pretty good.

There are exceptions; some genes behave differently. Programs like *edgeR* have methods to infer a common dispersion, with additional gene-specific ("tagwise") dispersion terms, and ways of combining them ("shrinkage toward the common dispersion"). Again for the sake of seeing the main ideas, we'll skirt past that additional complexity.

Ideally, as Bayesians, we want to marginalize over the unknown ϕ, but it's easier to make a point estimate for it. *edgeR* uses a particular technique that they call "quantile-adjusted conditional maximum likelihood" (qCML) to estimate a common dispersion ϕ. It's not entirely straightforward because each sample i has a different library size N_i, which we also don't know. Again, we won't delve into this calculation in detail. The main point is, *edgeR* is going to make an optimized point estimate for ϕ, essentially by

an iterative EM-style, maximum likelihood-ish algorithm.

ta-da: *edgeR*'s likelihood model

So in summary:

- The input data are mapped read counts y_{gi} (not TPM/RPKM)
- *edgeR* normalizes the counts to account for unequal library size
- *edgeR* optionally normalizes again to account for indirect effects on relative abundance
- now *edgeR* has reweighted count data that it calls *pseudocounts*
- *edgeR* can then estimate a common dispersion ϕ across all genes and samples

This gives us pretty much the likelihoods we'd need to do some inference, for example of what the unknown means μ_g are, and whether they differ across conditions.

the edgeR statistical significance calculations

How do we use this to decide which genes are differentially expressed?

Now it turns out that *edgeR* is only going to worry about the hypothesis H_0, that all samples came from the same model. It's not particularly Bayesian; it's not going to explicitly model hypothesis H_1. *edgeR*, like many statistical tests, does "null hypothesis significance testing" (NHST). It returns its answers in terms of a **p-value** and a **false discovery rate (FDR)** per gene. We've just seen p-values and FDRs last week, and now we have an example of using them in practice.

Look at the output file of *edgeR* and you'll see columns for "PValue" and "FDR". The "PValue" is

the probability *per test* (per gene). The FDR is, if you took all m "differentially expressed" gene in the list from this one and up, how many of those top m are expected to be false positives. Usually, we're interested in the *set* of predictions and how good it is, so it's more often the FDR that we're wanting to look at, and to control.

FDR calculation from the p-value

Getting to the FDR from the p-value is straightforward. Suppose we've ranked all n genes by their p-value, and we set a cutoff threshold at the r'th best gene -- i.e. we take the top r genes and call them "differentially expressed". Let the p-value of the r'th gene be p_r; then we expect up to np_r false positives with this p-value or better (because the p-value is literally the false positive rate). We made r predictions, and we expect up to np_r to be false positives... the **false discovery rate** (FDR) is the fraction of the r predictions that we expect to be false, $< \frac{np_r}{r}$.

The name for this is the *Benjamini-Hochberg procedure*.

People typically choose FDR thresholds of 0.05 or so. This means that if you called 100 genes differentially expressed at FDR< 0.05, about 5 genes really aren't.

The more technically involved calculation is how *edgeR* gets the p-value.

p-value calculation

The *edgeR* documentation is somewhat vague about how p-values are calculated:

> .... we can proceed with testing procedures for determining differential expression using the exact test. The exact test is based on the qCML methods. Knowing the conditional distribution for the sum of counts in a

> group, we can compute exact p-values by summing over all sums of counts that have a probability less than the probability under the null hypothesis of the observed sum of counts. The exact test for the negative binomial distribution has strong parallels with Fisher's exact test....

The literature reference for the method is Robinson & Smyth (2008), but it's also vague: *"One can construct an exact test similar to the Fisher's exact test for contingency tables but replacing the hypergeometric probabilities with NB."*

Can we dig into this and understand it? Clearly the place to start is with **Fisher's exact test**, which is actually quite beautiful.

Fisher's exact test

A wide range of experiments aim to ask whether some event happens with a different probability in one condition versus another. Maybe we're interested in whether read counts for our favorite gene g are occurring with different probabilities in wild-type versus mutant samples; maybe we're interested in the number of failures in parts made in our factories in Boston versus Vancouver. The event (a read maps to our gene; a part fails) is *categorical*: either it happens or it doesn't. The conditions are also categorical. Experiments like these can be represented as integer counts in **2x2 contingency tables**:

	condition 1	**condition 2**	**marginal row sum**
event	**x**	r-x	**r**
nonevent	n-x	m-(r-x)	m+n-r
marginal col sums	**n**	**m**	m+n

Given data like these, we want to ask whether the data support a conclusion that the event probabilities in the two conditions are different. The *null hypothesis* is that they're the same: the probability of an event is just p in both conditions, and any difference between the observed frequencies in the two conditions is just sampling noise.

If we assume that these counts are binomially distributed -- if the data are behaving like we're drawing balls from an urn, with probability p of a white ball -- then we know how to calculate the probability of count data given p. For example,

$$P(x) = \binom{n}{x} p^x (1-p)^{n-x}.$$

We could then construct a p-value test by declaring a suitable definition of "more extreme values of the data". For example, if x seems on the low side, we could calculate a one-tailed p-value for $P(X < x \mid p)$.

But the tricky bit is that *we don't know* p; the whole point is that we're trying to infer stuff about what the unknown p is, given the observed data.

Fisher believed (emphatically) that it is invalid to talk about a probability of p. For him, the Bayesian approach of marginalizing a likelihood of a hypothesis over the uncertainty of p was anathema. He needed to make the uncertainty of p *go away completely*. Beautifully, and counterintuitively, in this case there's a way to make that happen.

I wrote the 2x2 contingency table in a particular way, showing its marginal row and column sums, and defining all 9 count values in terms of four numbers: the total observations n and m for the two conditions (marginal column sums), the total number of events r (marginal row sum), and the number of events x in condition 1.

Suppose we phrase our question this way: given

that we observe r total events in the two conditions, what is the probability that they partition into x in condition 1, and $r - x$ in condition 2? This is a *conditional* probability. Let's define random variables X and Y for the observed event counts in condition 1 vs. 2. Then we're interested in $P(X = x \mid X + Y = r)$, which is:

$$P(X = x \mid X + Y = r) = \frac{P(X + Y = r \mid X = x)P(X = x)}{P(X + Y = r)}$$

Crucially, $P(X + Y = r \mid X = x)$ -- the probability that we get a total of r events out of X and Y, given that X has generated x events -- is just $P(Y = r - x)$, because X and Y are independent. So:

$$P(X = x \mid X + Y = r) = \frac{P(Y = r - x)P(X = x)}{P(X + Y = r)}.$$

Plugging binomial distributions in:

$$P(X = x \mid X + Y = r) = \frac{\binom{m}{r-x}p^{r-x}(1-p)^{m-r+x} \cdot \binom{n}{x}p^{x}(1-p)^{n-x}}{\binom{m+n}{r}p^{r}(1-p)^{m+n-r}}$$

All the terms involving p cancel (wtf!!!), leaving us:

$$P(X = x \mid X + Y = r) = \frac{\binom{m}{r-x}\binom{n}{x}}{\binom{m+n}{r}}$$

This has a name: the *hypergeometric distribution*.

To get to a p-value, because we assume a fixed row sum r, the "more extreme data sets" are those with a partitioning of r that's more extreme: like, if x is lower than its expectation under the null, we sum hypergeometric probabilities over all data sets with x or fewer events.

This p-value is called **Fisher's exact test**. It applies to 2x2 contingency tables of binomially distributed data. The null hypothesis is that event counts are being sampled with equal probabilities p in the two conditions. The trick is to assume that

all possible alternative outcomes are constrained to have identical marginal column and row sums: in particular, that the total number of events is r, and we're testing whether the partition into x and $r - x$ counts in the two conditions is within what we'd expect from sampling noise around expectations of np and mp.

extension to multiple replicates and the negative binomial

How do we get a 2x2 contingency table when we've done a bunch of biological replicates for our two conditions, for a bunch of genes?

First, we test each gene one at a time. The "event" is a mapped read count of gene g, compared to all other counts n_i in sample i.

Second — interestingly — the only thing that edgeR uses biological replicates for is when it estimates the common dispersion ϕ. Once it's done that, it calculates the *sums* of the counts in condition 1 versus 2: $y_{gc} = \sum_i \delta(c_i = c) y_{gi}$. The justification is that the distribution of two negative binomial random variables $X_1 \sim NB(\mu_1, \phi)$, $X_2 \sim NB(\mu_2, \phi)$ with a common dispersion is $X_1 + X_2 \sim NB(\mu_1 + \mu_2, \phi)$.

Thus for each gene, it constructs a 2x2 contingency table, for the observed count sums y_{gc} in the two conditions $c = \{0, 1\}$. It assumes a fixed marginal row sum $y_{g0} + y_{g1}$, and tests the null hypothesis that the partition of this sum into y_{g0} and y_{g1} is reasonable, given the expected means $p_g(\phi)n_0$ and $p_g(\phi)n_1$ for total summed sample sizes n_0 and n_1 in the two conditions, and an (unknown) negative binomial probability $p_g(\phi)$ for obtaining a read that maps to gene g, given the common dispersion ϕ.

Unfortunately, if we apply Fisher's conditional probability trick to the negative binomial,

unknown probability terms don't conveniently cancel. We don't know the negative binomial probability unless we also know a mean μ_g, in addition to the common dispersion. The unknown μ_g stays in the equation, making Fisher spin in his grave. Instead, *edgeR* has to make a point estimate of the mean expression μ_g under the null hypothesis: $\frac{y_{g0}+y_{g1}}{n+m}$. This might work fine, but it seems to defeat the beauty of the Fisher exact test. So although *edgeR* calls the test "exact", I'm not so sure it is!

This point estimate for the mean expression of g is only going to be reasonably accurate if we have plenty of counts y_g. If it's going to fail, it will fail on genes with small numbers of counts. I don't think it's a coincidence that the *edgeR* manual strongly recommends a filtering step to remove genes with small numbers of counts.

Mind you, though, I've reconstructed this from guesswork and perusing some of the *edgeR* source code. If an *edgeR* expert wants to correct me, please do! I've also simplified the above description, leaving out *edgeR's* pseudocount calculations that normalize for unequal library sizes.