week 09: maximizing your expectations

reading for this week

- Strengthened by weeks of Python and probabilistic inference, full of confidence, we're going to pull a bunch of pieces together. This week is a big week - sort of a review, sort of a midterm, sort of a summing up. We know enough now to understand the heart of an important RNA-seq data analysis paper, Li, Ruotti, Stewart, Thomson, and Dewey (2010), *RNA-seq with read mapping uncertainty*. We're prepared to understand its Methods section - the generative model in section 2.1, and the expectation maximization algorithm in section 2.2.

- We've already been warmed up on expectation-maximization algorithms in our week on k-means clustering. Now we see EM again in a different context. To really hammer it home, in the Wednesday lecture, we're going to work through yet another EM problem in biological data analysis: the problem of identifying a small consensus DNA motif sequence shared by a set of long-ish unaligned sequences. Some relevant background there includes *Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment*, Lawrence et al (1993), and *Identifying protein-binding sites from unaligned DNA fragments*, Stormo and Hartzell (1989).

probabilistic graphical models

Figure 1 from Li et al. (2010) is a **probabilistic**

graphical model, also known as a **Bayesian network**. A lot can be said about probabilistic graphical models -- one good doorstop of a textbook is from Koller and Friedman -- but we only need to know some basic essentials right now, and the basic essentials are straightforward.

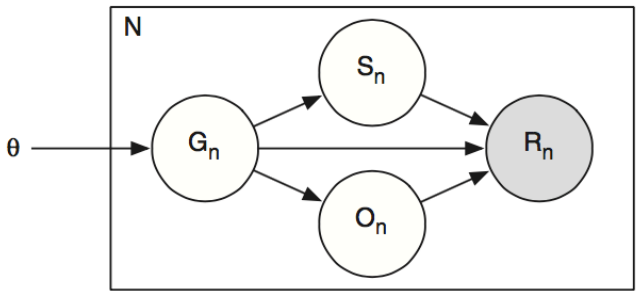


Fig. 1. The graphical model for RNA-Seq data used by our method.

The directed probabilistic graphical model from Li et al (2010). "Directed" means there are arrows representing dependencies as conditional probabilities. as opposed to "undirected" graphs which only show dependencies as connections (undirected edges).

We use a graphical model as an intuitively powerful way to state our assumptions about what the random variables are, and where there statistical dependencies are (and aren't) in an inference problem. Specifically, in a so-called **directed graphical model**, we use nodes (random variables) and arrows (directed edges, representing conditional probability dependencies) to think about how we're going to start with an (often nastily complex) joint probability distribution over all our random variables, and factor that down into a product of conditional probability distributions that we can work with.

the biological problem: ambiguous read mapping

Let's start with the specific example of Li et al. (2010). First let's state the RNA-seq analysis problem they're trying to deal with, in terms of how we imagine that the observed data are generated. That is, let's state our *generative probability model* for the RNA-seq data.

The problem that they're trying to tackle is that

just because I observe a read sequence R, I don't necessarily know which transcript isoform G to assign that read to. Transcripts can share identical stretches of sequence because of overlapping isoforms, or because of repetitive elements. **Multimapped reads** are reads that the read mapping software can't place uniquely, instead mapping them to two or more possible sources. Li and Dewey's framing distinguishes *gene multireads* that map to different genes (because of repetitive elements) from *isoform multireads* that map to different isoforms of the same locus (because of overlapping exons).

Making the problem worse, DNA sequencing isn't 100% accurate. We have to worry about similar matches, not just identical matches. Read mappers have to tolerate a small number of mismatches, typically one or two. Now a read might map to similar sequences, not just identical ones; and now we also have useful probabilistic information about the source. Depending on a sequencing error model that we could parameterize, we could guess that the read's true source is more likely to be more similar (or identical), compared to a less similar potential source. It's even possible, with enough sequencing error, that we could **mismap** a read only to false sources, and miss the true one.

the generative model

Li et al. state their generative model for one observed read n as follows.

- There are M possible transcript isoforms, $G = 1..i..M$, with true expression abundances $\theta_1 .. \theta_M$. These are *nucleotide abundances*, what we've also called ν_i before, as opposed to *transcript abundances* τ_i in TPM.
- Select a transcript $G = i$ with probability θ_i.
- Given transcript $G = i$, which has a length

L_i, select a starting point for a fixed-length sequencing read, $S = 1..j..L_i$. This probability could be uniform $\frac{1}{L_i}$. A more sophisticated model could try to capture the fact that read positions are known to be nonuniformly distributed because of end effects in making libraries (fewer reads at the 5' and 3' ends), and sequence composition biases (GC-rich reads are often underrepresented, and standard cDNA synthesis protocols use a specific nonrandom pool of oligonucleotide primers). Li et al. call this a *read start position distribution*. We'll just stick with a uniform distribution for our discussion here. Li et al also do some hand-waving to deal with an edge effect, that not all start positions can give you a complete read: they say, hey, there's an added poly-A tail that's typically longer than the read length, so even $j = L_i$ works as a start position (um... not really, the way many library prep kits work, but we'll go with it).

- In strand-nonspecific RNA-seq protocols, we're going to observe this sequence in either orientation (forward/reverse, Watson/Crick); while in strand-specific RNA-seq protocols, we're usually going to observe this sequence only in one orientation. So Li et al. say that we sample an orientation O for the read, a binary value 0 or 1. (They make this depend on G (i.e. $P(O \mid G)$), because of a detail in their model that I'm glossing over, that they include a "transcript" $i == 0$ that models unmapped reads, and for an unmapped read the orientation is irrelevant. For our purposes, we could just think of $P(O)$ independent of G.)

- Given a true sequence S and an orientation O, we generate an observed read R by modeling sequencing error. For example, if our sequencing process gives a 1% error rate

per position, we'd generate read R by dirtying up S with 1% error at each position. Li et al generalize this, and imagine a matrix $w_t(a, b)$, the probability that we see residue a at position t of a read, if a corresponds to a true residue b; the probability of the read R is a product of these individual terms.

the random variables

Thus we have the following random variables:

- θ, the expression levels, as nucleotide abundances (real-valued probabilities between 0 and 1; $\sum_i \theta_i = 1$)
- G, the transcript identity (takes value $i = 1..M$ for M different isoforms)
- S, the sequence read start position (takes value $j = 1..L_i$)
- O, the orientation (takes value $0 \text{ or } 1$)
- R, the observed read sequence (a sequence $x_1 . . x_\ell$, for fixed read length ℓ

In any modeling problem, some of our variables are **observed data**, some of our variables are **unknown model parameters**, some are **fixed model parameters**, and some are **hidden** (or **latent**) variables that represent the process by which the model generates observed data, but are not directly observed themselves.

Here, the read sequences R are observed data. θ are the unknown model parameters. The process that gets us to observed data requires specifying three hidden variables: the transcript source G, the start position S, and the orientation O.

The transcript lengths L_i are an example of fixed model parameters that we assume are known. We could keep them in our conditional probability expressions below, to be very explicit, but for conciseness, we're not going to.

the likelihood of the model

The **likelihood of the model** is the probability that the model generates observed data R, given the model parameters θ:

$$P(R \mid \theta)$$

but with our hidden variables in play, what we actually know how to write down is

$$P(R, S, G, O \mid \theta)$$

the joint probability of the data and the unknown hidden variables. To get the likelihood, we have to marginalize (sum out) the unknown hidden variables:

$$P(R \mid \theta) = \sum_S \sum_G \sum_O P(R, S, G, O \mid \theta)$$

Hidden variables are sometimes called **nuisance variables**: variables that have to be specified in order to calculate a likelihood as an actual number, but whose values are not relevant or interesting to us (at least at the moment), so we marginalize over their uncertainty to make them disappear.

factorizing and stating independence assumptions

If we literally needed to specify $P(R, S, G, O \mid \theta)$, we would have to specify (or at least imagine) an enormous multidimensional table of probabilities for all $4^\ell \cdot \sum_i L_i \cdot M \cdot 2$ possible combinations of outcomes R, S, G, O even if we only had just *one* choice of θ. Then we'd have one of those tables for each possible choice of θ. Since θ is a vector of continuous real-valued probabilities, we'd have an infinite number of tables. This is not good.

This is why we stated the generative model as a succession of steps:

- given θ, choose $G = i$ with probability θ_i;

- given $G = i$, choose $S = j$ with probability $\frac{1}{L_i}$ (say);
- given $G = i$, choose $O = k$ with probability $\frac{1}{2}$ (say);
- given $G = i, S = j, O = k$, generate R using the sequencing error model.

So, let's do this. Without making any assumptions (yet), we can correctly factorize our joint probability into a product of conditional probabilities in different ways, including one that corresponds to the order that we're thinking of our generative process in:

$$
\begin{aligned}
& P(R, S, G, O \mid \theta) \\
&= P(R, S, O \mid G, \theta)\ P(G \mid \theta) \\
&= P(R, O \mid S, G, \theta)\ P(S \mid G, \theta)\ P(G \mid \theta) \\
&= P(R \mid O, S, G, \theta)\ P(O \mid S, G, \theta)\ P(S \mid G, \theta)\ P(G \mid \theta)
\end{aligned}
$$

This gives us our joint probability in terms of a product of conditional probabilities of our individual random variables.

Now we can walk through each conditional probability term and formally state our **independence assumptions**, which make our model tractable. From right to left:

- The probability of selecting isoform G given θ is, well, $P(G \mid \theta)$, so keep that.

- The probability of selecting a read start position S only depends on the length L_i, hence on the choice $G = i$. It's independent of θ. So $P(S \mid G, \theta) = P(S \mid G)$.

- The probability of selecting orientation O is independent of θ and S, so $P(O \mid S, G, \theta) = P(O \mid G)$. (Or even just $P(O)$, more simply.)

- The probability of generating read R is independent of θ, so $P(R \mid O, S, G, \theta) = P(R \mid O, S, G)$.

Now look at the Li et al Figure 1 again. Its nodes and arrows show exactly these conditionings (after dropping the conditioning on fixed model parameters like L because they're fixed). The label on a node is the stuff to the left of a $|$, the probability distribution of this random variable; the arrows that come into the node are the stuff to the right, the conditional dependencies. A directed graphical model is just a convenient pictorial representation of the conditional dependencies we want to assume in our model.

inference

Given a probability model, there are lots of things I might want to do with it. For example, given observed data R, I might want to know the posterior distribution of the unknown parameters θ. I'd write down Bayes' rule for getting the posterior distribution $P(\theta \mid R)$, in terms of the marginal likelihood. I'd be forced to specify a **prior probability distribution** $P(\theta)$ to do this.

If we could specify the prior *and* brute force enumerate or sample the space of all possible parameters θ (like in the Student's game night pset, where we used coarse-grain discretization of two parameters μ and θ), we could get this posterior probability distribution by calculating the likelihood times prior for all possible θ choices and normalizing. But here θ is an M-dimensional probability vector, for M on the order of tens of thousands, the number of different transcript isoforms in the transcriptome. We're not going to enumerate all the choices, even if we did assume a coarse-grained discretization of some sort. Even supposing we did some sort of binary discretization like high/low, there'd still be 2^M possible settings for θ we'd have to do.

So this posterior inference problem is still a high class problem for us -- buzzwords that we'll see for solving problems like that are things like **Gibbs sampling** and **Markov chain Monte Carlo** -- and

we're going to sneak up on it instead. Suppose that instead of getting the posterior distribution for θ, I would be happy to just make a point estimate for an optimal θ.

The first thing we have to worry about is, what do you mean *optimal*?

maximum likelihood parameter estimation

We've already seen that a standard definition of *optimal* is **maximum likelihood**: find the values of parameters θ that maximize the likelihood $P(D \mid \theta)$ for observed data D.

Suppose I told you the count c_i of reads that truly map to transcript i. Then you'd simply estimate $\theta_i = \frac{c_i}{N}$. That's the maximum likelihood estimate for θ_i -- but I gave you the counts c_i, which means I had to give you the hidden values of G_n for all the reads $1..n..N$. If we want to be super formal we can write these unknown "true counts per transcript" as:

$$c_i = \sum_n \delta(G_n = i)$$

where $\delta()$ is a so-called **Kronecker delta function**, often seen as a notational trick in counting subsets of things: it takes the value 1 when its condition is true, and 0 when its condition is false. Here, "count all the reads that are assigned to transcript i; that sum is c_i".

Your problem is that you don't know c_i, because the values of the G_n are hidden and unknown.

inference of hidden variables

But, given observed data R, and *if I told you* θ, you could *infer* G. By manipulating the likelihood $P(R, S, G, O \mid \theta)$, you can solve for G in terms of R:

$$P(G \mid R, \theta) = \frac{P(R, G \mid \theta)}{P(R \mid \theta)}$$

where the denominator there is just a renormalization, a marginal sum over the numerator, $P(R \mid \theta) = \sum_G P(R, G \mid \theta)$:

$$P(G \mid R, \theta) = \frac{P(R, G \mid \theta)}{\sum_G P(R, G \mid \theta)}$$

Alas, we don't know how to put numbers to $P(R, G \mid \theta)$; our model is such that we can put numbers to $P(R, G, S, O \mid \theta)$, in terms of additional nuisance variables S (the read start position) and O (the orientation). We get our likelihood term by marginalization over the nuisance variables:

$$P(G \mid R, \theta) = \frac{\sum_S \sum_O P(R, G, S, O \mid \theta)}{\sum_G \sum_S \sum_O P(R, G, S, O \mid \theta)}$$

So the computation would be to just calculate the probability $\sum_S \sum_O P(R, G = i, S, O \mid \theta)$ for each possible assignment of observed read R to transcript i -- now you have a vector of M likelihoods -- and you renormalize them to get the distribution $P(G \mid R, \theta)$.

This is your uncertainty about the assignment of observed read R to underlying transcript G. Now what you could do is assign each transcript G a fraction of a counted read, weighted by that uncertainty. You'd be calculating the **expected value** of the counts c_i, given your uncertainty about the mapping:

$$c_i = \sum_n P(G_n = i \mid R_n, \theta)$$

And then you could renormalize the c_i to get your maximum likelihood estimate of the θ.

Which is all just great, except that you had to know θ in the first place to get your expected counts c_i, but that's what we're trying to estimate!

expectation maximization

We have a chicken and the egg problem. If I told you the unknown c_i, you could estimate the unknown θ_i by maximum likelihood. If I told you the unknown θ, you could estimate the unknown c_i by probabilistic inference from the observed read data R. But both the c_i and θ are unknown.

Amazingly enough, the following apparently dumb solution works:

- Initialize a starting guess θ to anything (even random)
- Iterate:
- (Expectation:) Infer expected counts c_i given current θ.
- (Maximization:) Calculate updated θ given c_i.
- until converged.

As we've seen before with in k-means week, this is an Expectation-Maximization algorithm, or EM for short. It was formally and generally introduced in a famous paper by Dempster, Laird, and Rubin (1977), but as you might imagine from its apparent simplicity, it had shown up in various guises before.

Generally, EM is a local optimizer. For many problems that it is applied to, different starting guesses may yield different optima, if the posterior probability surface for $P(\theta \mid \text{data})$ is rough. It happens that in this case, Li et al. (2010) prove that there is only a single optimum for θ ("we prove that the likelihood function for our model is concave with respect to θ", in the jargon), so EM reaches a global optimum.

motif identification by expectation maximization

In the Li (2010) problem, we have three latent variables (G, S, O). We only need to infer counts for one of them (G). We have to push S, O around

and get them marginalized out. In the homework problem this week, we work with a more abstracted version of the problem that gets rid of the orienttation O and the sequencing error model $P(R \mid S)$, reducing the problem to a single hidden variable (the unknown transcript isoform G that a read R came from). We saw that k-means clustering was also in terms of a single hidden variable (the unknown cluster assignment for a data point). To beat this horse even more thoroughly, it might be useful to see another example of an important biological data analysis problem where there's only one hidden variable other than our observed data x and our unknown model parameters θ.

Here's the most basic case of the **motif identification** problem: we're given N DNA sequences $x^1 \ldots x^n \ldots x^N$. Each individual DNA sequence x^n is a string of L residues, $x_1^n \ldots x_i^n \ldots x_L^n$.

Embedded somewhere in each DNA sequence is exactly one **motif**, of length W. A motif might be a protein binding site, for example. A DNA-binding protein prefers certain sequences, and many DNA-binding proteins, including transcription factors, have distinctive site-specific preferences for individual residues: for instance we might have a protein that likes GaattCC, where it cares a lot more about the G and CC than it cares about the aatt.

W is short: typically 6-10 residues for typical DNA-binding proteins. L is long-ish: experiments like ChIP-seq experiments (where we recover DNA fragments bound to a protein of interest, and sequence the bound fragments) might give us sequences of length $L = 200 - 400$.

The game is to identify the W-length motif that all the sequences share, even though the motif is a probabilistic pattern of preferences, not merely the same identical subsequence everywhere.

A good approximate model of the preferences of a DNA-binding protein is a **position-specific weight matrix (PWM)**. The PWM is a generative probability model of a sequence of length W. In a PWM, we assume that each residue is generated independently, with probabilities for A/C/G/T at that position, i.e.:

$$P(y_1 \ldots y_W \mid \theta) = \prod_j p(y_j \mid \theta)$$

so therefore, the PWM θ consists of Wx4 parameters $\theta_j(a)$, 4 parameters for the probability of getting A,C,G, or T at each position $j = 1..W$.

We can specify the unknown location of the motif y in each larger sequence x by its start position λ. The possible start positions for λ range from $1..L - W + 1$ (there's an edge effect; you can't start too close to the end and have a complete motif). So each sequence x is composed of three segments: the first $x_1 \ldots x_{\lambda-1}$ are random, the motif $x_\lambda \ldots x_{\lambda+W-1}$ is a sample from the PWM, and the remaining $x_{\lambda+W} \ldots x_L$ are random.

If we wanted to be super compulsive, we could now state the probability of generating an observed sequence x, by stating that the random (non-motif) residues in it are generated with probability 0.25 for each base A,C,G,T. Then:

$$P(x_1 \ldots x_L \mid \theta, \lambda) = \prod_{i=1}^{\lambda-1} 0.25 \prod_{j=1}^{W} \theta_j(x_{\lambda+j-1}) \prod_{i=\lambda+W}^{L} 0.25$$

but you'll notice that there are always $L - W$ random non-motif residues, so really this is just a constant times the probability that the PWM assigns to the possible motif starting at position λ:

$$P(x_1 \ldots x_L \mid \theta, \lambda) = 0.25^{L-W} \prod_{j=1}^{W} \theta_j(x_{\lambda+j-1})$$

and we can even drop the constant and just deal

with a proportionality, since it'll turn out that we'll be renormalizing the likelihood for other reasons anyway:

$$P(x_1 . . x_L \mid \theta, \lambda) \propto \prod_{j=1}^{W} \theta_j(x_{\lambda+j-1})$$

So there's our data likelihood for one sequence x: the probability of x that contains one motif at position λ, given the PWM θ. The likelihood for N independent sequences x^i is the product over each.

If I told you the unknown λ^i for a data set x, you could extract the individual motifs, align them (put them in register, in columns 1..W), and collect the observed counts of residues $c_j(a)$ for each motif position j and each residue a. Given those counts, your maximum likelihood estimate for the PWM θ is just to use the residue frequencies at each position:

$$\theta_i(a) = \frac{c_i(a)}{\sum_b c_i(b)}$$

That is, if you find the motifs in 100 sequences, and at motif position 1 you see 80 A, 10 C, 10 G, and 0 T, then you estimate probabilities $\theta_1 = \{0.8, 0.1, 0.1, 0\}$.

If instead I told you the unknown PWM θ, then you could calculate your expected counts $c_i(a)$ via inferring the unknown position of the motif in each sequence:

$$P(\lambda \mid x, \theta) = \frac{P(x \mid \lambda, \theta) P(\lambda \mid \theta)}{P(x \mid \theta)}$$

$P(x \mid \lambda, \theta)$ is the data likelihood, which we already stated above. $P(\lambda \mid \theta)$ is the prior for λ, and we can state that a priori (before we've seen any sequence x, all positions $1..L - W + 1$ are equiprobable ($\frac{1}{L-W+1}$), thus constant, and this constant is present in both the numerator and all terms of the denominator, so it disappears.

(Similarly the proportionality constant 0.25^{L-W} we talked about for our likelihood is going to disappear.) The denominator $P(x \mid \theta)$ is just the sum of the numerator over all choices of position λ. So:

$$P(\lambda \mid x, \theta) = \frac{P(x \mid \lambda, \theta)}{\sum_{\lambda'=1}^{L-W+1} P(x \mid \lambda', \theta)}$$

which means, in pseudocode-ish words:

- For each position λ, calculate the likelihood given that this is the motif's true position: $L_\lambda = \prod_{j=1}^{W} \theta_j(x_{\lambda+j-1})$
- Sum L_λ over λ and renormalize.

Given a posterior distribution over the uncertain motif location λ^n for each observed sequence x^n, you collect expected counts by apportioning one total motif count per sequence across the possible λ according to the uncertainty:

$$c_i(a) = \sum_{n=1}^{N} \sum_{\lambda^n=1}^{L-W+1} P(\lambda^n \mid x^n, \theta)\ \delta(x^n_{\lambda^n+i-1} = a)$$

which may look harder than it really is, with that Kronecker delta in the notation, and those n superscripts. In words:

- for each sequence x^n:
- calculate the posterior distribution $P(\lambda^n \mid x^n, \theta)$ as above;
- for each possible motif position $\lambda^n = 1..L - W + 1$ in the sequence:
    - for each column i of the PWM:
        - Let a be the residue you see at position i of the PWM, when you've got the PWM aligned at λ^n.
        - Add a fraction of a count, $P(\lambda^n \mid x^n, \theta)$, to $c_i(a)$.

The EM algorithm is now essentially the same as we saw above for the RNA-seq inference problem:

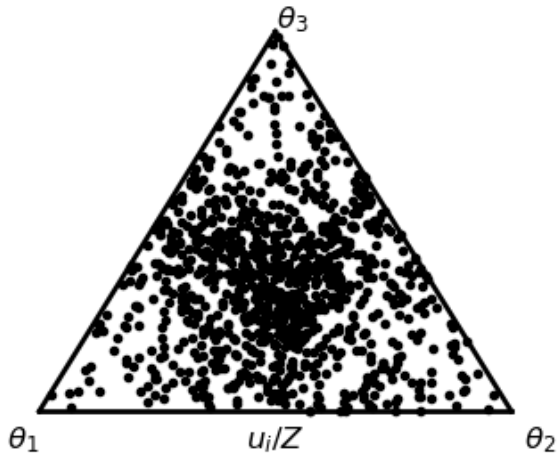
- Initialize a starting guess at the PWM θ to

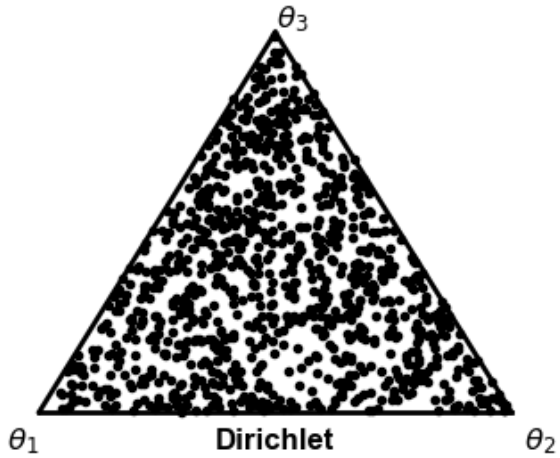
anything (even random)
- Iterate:
- (Expectation:) Infer expected counts $c_i(a)$ for the motif, given current θ.
- (Maximization:) Calculate updated θ given $c_i(a)$.
- until converged.

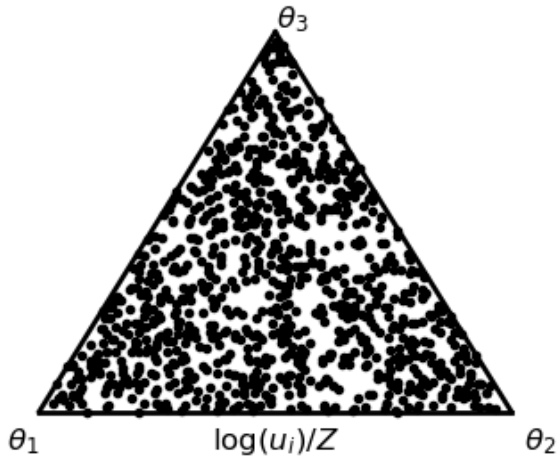
bonus: sampling random probability vectors from the Dirichlet

An interesting probability sampling problem arose when we said things like "initialize the starting θ to something random". We don't *need* to sample our starting guess uniformly for the Li et al (2010) EM algorithm -- any guess will do -- but suppose we did want to make a completely random guess at θ? How do we choose a uniformly random multinomial probability vector $\theta_1 . . \theta_K$, for size K? We're talking about sampling from a specified *probability distribution over probability vectors*, $P(\theta)$. We're going to need to think about such things, because in some Bayesian inference problems, we're going to want to specify *nonuniform* priors over model probability parameters. Thinking about how to sampling from a *uniform* $P(\theta)$ is one place to get started.

You might think you could just sample each individual p_i uniformly on 0..1 (using Python's `np.random.uniform`, for example) -- i.e. $p_i == u_i$, for a uniform random variate $u_i = [0, 1]$, but of course that doesn't work, because they're constrained to be probabilities that sum to one: $\sum_i p_i = 1$.







One bad way and two good ways to sample probability vectors uniformly. The examples show 1000 samples, plotted in so-called "barycentric" coordinates (also known as a "ternary plot"), to show the 3D simplex in 2D. The top panel (the bad way) shows sampling by drawing uniform variates and renormalizing. The middle panel shows sampling from a Dirichlet distribution with all alpha=1. The bottom panel shows sampling

uniform variates, taking their log, and renormalizing.

So then you might think, fine, I'll sample elements uniformly then renormalize the vector: $p_i = \frac{u_i}{\sum_j u_j}$. But this doesn't sample probability vectors uniformly, as shown by the clustering of samples in the middle of in the top panel of the figure to the right.

The right way to do it is to use the **Dirichlet distribution**. The Dirichlet is a distribution over multinomial probability vectors:

$$P(\vec{p} \mid \vec{\alpha}) = \frac{\prod_{i=1}^{K} p_i^{\alpha_i - 1}}{Z(\vec{\alpha})}$$

where the normalization factor $Z(\vec{\alpha})$ is:

$$Z = \frac{\prod_i \Gamma(\alpha_i)}{\Gamma\left(\sum_i \alpha_i\right)}$$

With all $\alpha_i = 1$, all probability vectors are sampled uniformly, because the $p_i^{\alpha_i - 1}$ term becomes a constant $p_i^0 = 1$ for all values of p_i.

So if you go over to Python's `np.random.dirichlet`, you'll find that you can do:

```
import numpy as np
theta = np.random.dirichlet(np.ones(K))
```

and you'll get a uniformly sampled probability vector θ of size K. That's illustrated in the middle panel to the right.

Then it turns out that in the special case of all $\alpha_i = 1$, you can prove that sampling uniformly from the Dirichlet is equivalent to taking the logarithm of uniform variates and renormalizing,

$$p_i = \frac{\log(u_i)}{\sum_i \log(u_i)}$$

or in Python

```
import numpy as np
theta = np.random.exponential(size=K)
Z     = np.sum(theta)
theta = np.divide(theta, Z)
```

This special case might be useful to know, like if you're programming in a language other than Python that doesn't put a `np.random.dirichlet` so conveniently at your fingertips. The bottom plot of the figure to the right shows using the $\log(u_i)/Z$ method.

If you want to play around with this, you can download the script I used to generate the figure.

The Dirichlet distribution will turn out to be useful to us for other reasons than just sampling uniformly-distributed probability vectors, as the next section delves further into.

a bit more on the Dirichlet

David MacKay often talked about the Dirichlet using an analogy to a "dice factory": a Dirichlet $\alpha_1 . . \alpha_6$ generates probability vectors $p_1 . . p_6$ that you could use as parameters of a die roll.

The Dirichlet appears a lot in probabilistic inference when we're working with likelihoods that are multinomial distributions (rolling dice, flipping coins, Laplace counting girl versus boy births in Paris, choosing an RNA isoform i by its abundance in an RNA population) and we want to specify a prior. If we've got multinomial probability parameters θ that generated observed counts c, and we want to specify the joint probability of everything $P(c, \theta)$, then we'll break that down into a likelihood $P(c \mid \theta)$ and a prior $P(\theta)$. The Dirichlet, with its parameters α, gives us a mathematically convenient way to parameterize a prior $P(\theta)$ as $P(\theta \mid \alpha)$.

There are other ways one might parameterize a prior $P(\theta)$. The advantage of the Dirichlet is that it plays well with the multinomial; it is the

multinomial's **conjugate prior**. The multinomial is:

$$P(c \mid p) = \frac{\prod_i p_i^{c_i}}{Z}$$

where the normalization constant Z is the reciprocal of the multinomial coefficient:

$$Z = \frac{\prod_i c_i!}{\left(\sum_i c_i\right)!}.$$

When you put that together with a Dirichlet prior to get a joint probability $P(c, \theta \mid \alpha) = P(c \mid \theta)P(\theta \mid \alpha)$ you get:

$$P(c, \theta \mid \alpha) = \frac{\prod_i p_i^{c_i + \alpha_i - 1}}{Z(\alpha, c)}$$

which is just as easy to handle as the multinomial alone when we need to do any integration or differentiation.

The binomial distribution is the special case of the multinomial, so you can also apply a Dirichlet prior to a binomial, and integrate the result using Beta functions just like Laplace integrated the binomial by itself.
