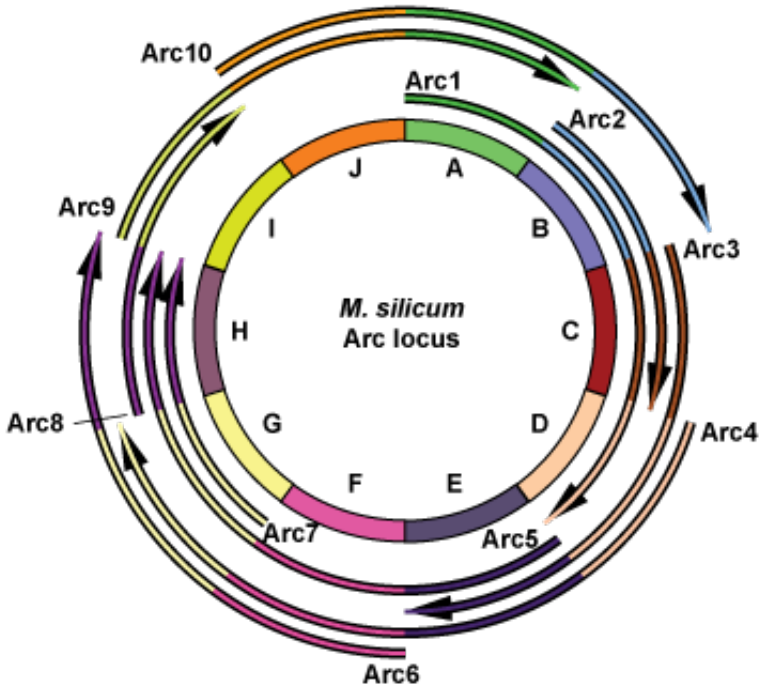
pset02: the adventure of the ten Arcs

The sand mouse genome has an unusual locus called *Arc* that expresses ten overlapping mRNA transcripts called Arc1 through Arc10.



The sand mouse *Arc* locus consists of ten segments A-J. Starting at each segment, it is transcribed into ten overlapping mRNA transcripts, Arc1-Arc10, each of which is 2-4 segments long.

The Arc mRNAs are circularly permuted, because the DNA at the *Arc* locus is excised from the genome and circularized. (Things like this do happen in biology -- for example, there are extrachromosomal circular ribosomal DNAs in the frog *Xenopus*.) The *Arc* locus consists of ten segments (labelled A through J). Each segment is exactly 1000nt long, so the total Arc circle is 10kb. Each of the ten Arc mRNA isoforms starts at a corresponding segment, and is 2-4 segments (2000-4000nt) long.

You've just published part of your PhD thesis research, in which you accurately measured the per-nucleotide abundances ν_i for all ten Arc transcripts (and converted them to TPM), and you concluded that Arc1 and Arc6 are the rarest transcripts, by a factor of about 3x:

transcript	abundance (ν_i)	TPM ($10^6 \tau_i$)	length (L_i)	segments_covered
Arc1	0.008	6000	4000	ABCD
Arc2	0.039	58000	2000	BC
Arc3	0.291	290000	3000	CDE
Arc4	0.112	83000	4000	DEFG
Arc5	0.127	94000	4000	EFGH
Arc6	0.008	7800	3000	FGH
Arc7	0.059	87000	2000	GH
Arc8	0.060	88000	2000	HI
Arc9	0.022	22000	3000	IJA
Arc10	0.273	270000	3000	JAB

Moriarty's experiment

Moriarty, a senior postdoc in the lab, doesn't think you know how to do RNA-seq experiments. He's done another RNA-seq experiment on the sand mouse *Arc* locus himself, under the same experimental conditions as you.

He did 100,000 single-ended, non-strand-specific 75nt reads from a cDNA fragment library of mean length 150bp, standard deviation 20bp, with an estimated base-calling accuracy of 99.9% per base (Q30 bases). He analyzed his data with kallisto. Based on his results, Moriarty says that the Arc1-Arc10 transcript abundances (in TPM) are actually more like:

transcript	TPM

Arc1	19000
Arc2	55000
Arc3	280000
Arc4	78000
Arc5	92000
Arc6	17000
Arc7	84000
Arc8	88000
Arc9	25000
Arc10	260000

and based on that, he says you underestimated the abundance of Arc1 by about 3-fold and Arc6 by about 2-fold -- so much so, that he challenges one of your main conclusions. His analysis says that Arc1 and Arc6 aren't as rare as you say, and that Arc9 is expressed about the same level as them.

There are no obvious differences in how Moriarty did his RNA-seq experiment - the only difference seems to be that he used *kallisto* to do his quantitation, whereas you used your own independent mapping and EM quantitation code.

You decide to reproduce Moriarty's analysis, and to use a positive control simulation to test the accuracy of kallisto's quantitation.

1. use kallisto and reproduce Moriarty's result

Download and install kallisto:

- the kallisto web site
- the download page
- the instructions for building from source

kallisto runs on Mac OS/X, Linux, and other unix-like systems. To get kallisto running on Windows

10, you'll want to install Ubuntu bash.

Moriarty provides you with a gzip'ed FASTA file containing the transcript sequences of the 10 Arc transcripts, and a gzip'ed FASTQ file containing his 100,000 reads.

Use kallisto to analyze these data and reproduce Moriarty's result.

2a. simulate an Arc transcriptome and RNA-seq reads

Write Python code to simulate an Arc locus, an Arc transcriptome, and 100,000 reads from an RNA-seq experiment on the Arc transcriptome. Your script generates two simulated data files that kallisto will take as its input:

- a FASTA format file of the transcriptome
- a FASTQ format file of the reads

Your simulation will be controlled by several parameters. It'll be useful to leave them as settable parameters that you can play with in different simulations. For example, here's a chunk from mine:

```
# Set up the Arc locus
#
S         = 10           # Number of segments in the
T         = S            # Number of different transc
N         = 100000       # total number of observed r
len_S     = 1000         # length of each segment (nu
len_Arc   = len_S * S    # total length of the Arc lc
len_R     = 75           # read length
```

Set up your simulation so that it can either use the structure of the Arc locus shown above (i.e. those specific lengths L_i and those specific abundances ν_i), or it can generate new random choices for lengths of each transcript (in segments) L_i and abundances ν_i.

To generate an Arc locus DNA sequence:

- assume S=10 segments, A-J
- assume each segment is 1000 bp long
- make random DNA of uniform base composition, 25% each base (ACGT)
- your total DNA sequence will be 10,000 bp

To generate the Arc1..Arc10 mRNA transcripts:

- extract them from the Arc locus by their coordinates (being careful to handle the circular permutation!)
- write them in FASTA format to a `.fasta` output file.

To generate reads, let's simplify things so we don't have to worry about different fragment lengths, read orientation, or base calling error, and you can just do:

- sample a transcript i according to its nucleotide abundance ν_i;
- pick a 75nt read by choosing a random start position in the transcript (remember that only positions 0..L-75 are valid start positions for a read of length 75).
- output the read in FASTQ format to your read file. Set the basecall quality code to 'I', as in the test data files provided with kallisto. Repeat for all 100K reads.

Because you didn't simulate the process of obtaining a cDNA fragment, and instead just directly took a 75nt read from the transcript, you'll need to set the fragment length parameters for `kallisto quant` to be suitable for fragment lengths of about 75nt in single-ended data. Set the `-l` fragment mean length parameter to 75 and the `-s` fragment length std dev parameter to something small but greater than 0, as in:

```
kallisto quant -i <indexfile> -o <outputdir> --si
```

kallisto's abundance estimates turn out to be surprisingly sensitive to details of how the fragment length distribution is set, which I didn't realize until I made a simpler path through the pset. If you use `-l 150`

2b. alternative (harder)

version of part 2

instead of `-l 75` in the simple version of the pset, for example, `kallisto` estimates the abundance of Arc6 as near-zero. It's not clear why that should be so, from the published description of kallisto.

Optionally, if you're already a good Python coder and you'd like to do a more realistic and challenging simulation, you can do your simulation for part 2 including a fragment length distribution, read orientation (your read can come from either end of the fragment), and base call error. For example, my extra parameters for this are:

```
alpha     = 0.999        # base calling accuracy (Q30
mean_frag = 150          # fragment size: mean (of a
sd_frag   = 20           # fragment size: stdev
```

kallisto models the cDNA library fragment length distribution (so that it can calculate an "effective length" of each mRNA, correcting for the fact that library fragmentation and size selection selects against small cDNAs). So to generate each read, first have your simulation generate a random fragment, then generate a read from one of its ends:

- sample a transcript i according to its nucleotide abundance ν_i;
- pick a random fragment length at least as long as one read from a truncated Gaussian of mean length 150, standard deviation 20 (truncated because: no shorter than a read, and no longer than the whole transcript), i.e. something like:

```
while True:
    fraglen = int(np.random.normal(mean_frag, sd
    if fraglen >= len_R: break
if fraglen > L[i]: fraglen = L[i]
```

- pick a random fragment of that length from transcript i;
- pick a 75nt read by choosing a random orientation, taking the read from one of the two ends of the fragment. If on the top

strand, your read is the first 75nt of the fragment. If on the bottom strand, your read is the last 75nt of the fragment, reverse complemented.

- generate a simulated read sequence by adding simulated base calling errors to the 75nt read: given base calling accuracy α, at each base, with probability $(1 - \alpha)$, change it to one of the other 3 bases.

- output the 75nt read starting from that position in FASTQ format to your read file. Repeat for all 100K reads.

In this version, in your `kallisto quant` step, you'll give kallisto the same fragment length distribution parameters you used to simulate the data, i.e.:

```
kallisto quant -i <indexfile> -o <outputdir> --si
```

3. test kallisto

Use your simulator (from 2a or 2b above) to generate dataset(s) using the lengths L_i for the Arc locus as shown above, and the abundances ν_i that you think are true (first table, above). Analyze the simulated data with kallisto.

Does kallisto get the correct answer?

4. "debug" kallisto

Maybe you just found that kallisto isn't actually getting its estimation quite right.

Suggest a plausible reason that kallisto might be messing up the *Arc* analysis. What's most unusual about *Arc*, that might violate kallisto's assumptions somehow? Design at least one experiment that uses your simulator to test your hypothesis -- i.e., identify a modification that gives simulated data that kallisto works fine on.

turning in your work

Submit your Jupyter notebook page (a `.ipynb` file) to the course Canvas page under the Assignments tab. Please name your file `<LastName><FirstName>_<psetnumber>.ipynb`; for example, mine would be `EddySean_02.ipynb`.

Remember that we grade your psets as if they're lab reports. The quality of your text explanations of what you're doing (and why) are just as important as getting your code to work. We want to see you discuss both your analysis code, and your biological interpretations.

hints

- Conversion between "nucleotide abundance" units ν_i (in Li and Dewey's notation) versus "transcript abundance" units in TPM (τ_i) is a pesky detail you have to keep straight. Recall:

$$\tau_i = \frac{\nu_i}{\ell_i} \left(\sum_j \frac{\nu_j}{\ell_j} \right)^{-1}$$

and multiply by 10^6 to convert a fraction τ_i to "transcripts per million" (TPM). You can check for yourself that the `est_counts` per transcript in a kallisto `abundance.tsv` file are proportional to ν_i: if you divide by `eff_length`, renormalize, and multiply by 10^6 you get kallisto's TPM estimate.

- As far as I know, kallisto doesn't pay any attention to the quality values (QV's) in the FASTQ file. The example data that comes with kallisto uses a QV code of 'I' for every base; 'I' is ASCII character 73, 73-33=40, so that's a Q40 base, which means an error probability of 10^{-4}. (Why yes, I do have an ASCII code table pinned above my computer. Don't you?)