Introduction to Bioinformatics
Problem Set 8: Mixed Bag

1. If there's only one lesson you gain from this course, let it be a respect for the connection between evidence and assertion. Some of you may object, "Hey! That's the same thing you said last semester in *Molecular Biology*!", to which I respond, "Yes,... evidence of my own respect." With that in mind, I present to you the following scenario.

Suppose you are an editor at a high class scientific journal, and you are considering the following submission for publication:

> *Mary had a little lamb, whose fleece was white as snow,*
> *And everywhere that Mary went, the lamb was sure to go.*
> *It followed her to school one day, which was against the rules.*
> *It made the children laugh and play, to see a lamb in school.*

You're inclined to reject the manuscript out of hand. It is shorter than the usual submission to your journal, but the main objection is the lack of what you perceive to be the rigor a scientific article requires. But your heart softens, and you resolve to fulfill your calling as an editor by editing, rewriting the submission to meet your specifications. What follows is the first paragraph of what came of your effort, the **Results** section of an article in which ***every assertion is connected to an observation***, either yours or someone else's, and ***every observation is connected to the means by which it was produced***.

Cultural Impact of Human-Ovine Mutualism

Results

An adolescent human female (code-named "Mary") was tagged with a fluorescent protein for subsequent identification and then released. She was repeatedly observed (n=7) within 2 meters of a sheep with juvenile morphological characteristics [1] whose length from the ischium (pelvis bone) to the scapula (shoulder bone) was measured to be 18.7 cm. This is smaller than 95% of lambs in the 2010 U.S. lamb census [2].

. . .

References

1. Reese WO (2009). Functional Anatomy and Physiology of Domesticated Animals. Fourth edition. Wiley-Blackwell, Danvers MA.
2. U.S. Bureau of Agronomic Statistics (2010). Morphological Assessment of Livestock. p.47. U.S. Government Printing Office, Washington DC.

Your task now is to complete the **Results** section of the manuscript, following the original as closely as possible but adhering to your high standards regarding evidence. However, those high standards for some reason do not prevent you from making up any facts you may need and any references to justify those facts.

2. One of the most commonly used strategies in sequence analysis is the moving window, in which you consider small segments of a much larger sequence, one after the other. Here, you'll use this strategy to make your own DOTPLOT function. Dotplots are a means of comparing one sequence with another. They're discussed in the tour of Hatfull (Part I).

Your strategy to construct a dotplot will be to slide a window across one sequence and compare that window against the second sequence. The matches for each window will provide information for one line of the dotplot, using the PLOT function to display the dotplot. For example, to construct a dotplot of Sequence **A** vs Sequence **B** (see below), you'll compare the first 11 nucleotides of Sequence **A** to every point in Sequence **B**, noting where matches occur.

Sequence **A**: **CAACCCTCCAT**CATAAAACTTGGGCTTGGGAGGCAGAGCCTAACCTCTCT

Sequence **B**: CAACCCTCCATCATAAAACAACCCTCCATCATAAAACTTGGGCTTGGGAG

A match of the first window occurs in Sequence **B** at positions 1 and 20, so the first line will contain a dot at each of those positions. Then you slide the window one nucleotide to the right and repeat the comparison to obtain the dots for the second line, and so forth.

As a test of the function (and to give us raw material as we build it), we'll try it on the sequences of two small phage: Chlamydia phage 3 and Chlamydia phage phicpg1 (find the sequences in either Phantome/BioBIKE or ViroBIKE).

Your implementation of DOTPLOT will compare two sequences through a window of 11 nucleotides, grabbing 11 nucleotides at a time from the first sequence and searching the second sequence for matches to it. As you go through each step below, notice the result in the result pane and compare it with what you expect from the actual sequence.

2a. In the end you will use the PLOT function to display the results of your sequence comparisons. To see where you're heading, start off by experimenting with the PLOT function. One way to use PLOT is by providing it with a set of number pairs, such as `((1 1)(2 4)(3 9))`. Play with PLOT until you understand what it does with the pairs.

2b. DEFINE `seq1` as the SEQUENCE-OF Chlamydia phage 3 and `seq2` as the SEQUENCE-OF Chlamydia phage phicpg1.*

2c. DEFINE `window-size` as 11 (each comparison will be 11 nucleotides)

2d. DEFINE `start-of-window` as 1 (we'll start the first window at coordinate 1

2e. DEFINE `end-of-window` as… as what? If `start-of-window` is the first coordinate of the window, what is the last? Knowing the start-of-window and the window-size, how would you calculate the end coordinate? Keep in mind that the values of these variables may change. Use SUM-OF to do the calculation. The function will look something like this:

* Of course the names of the first phage will not be literally "Chlamydia phage 3". That's just the human name for the phage. Use the usual tricks to find the name of the gene as known to BioBIKE. By the way, NAME-OF [organism] SHORT will give you a short nickname,.

What value should go in the *number* box? Check this by counting!

2f. Now use the values you've defined to define the portion of the first sequence you'll try to match with the second sequence. DEFINE `window` as SUBSTRING `seq1` FROM `start-of-window` TO `end-of-window`

2g. DEFINE `matches` as MATCHES-OF-ITEM `window` IN `seq2`. It will be more convenient if the function returns only the position of the match, so specify the +COORDINATES option. Try it with and without the option so you can see what I mean.

2h. Now we need to turn this information into number pairs. The first number in each pair will be the coordinate of the window (for simplicity, we'll use `start-of-window`). The second number of each pair will be the coordinate of a match. A pair will be formed by:

LIST `start-of-window n`

where n is one of the coordinates returned by MATCHES-OF. We don't know how many matches will be found, but we can MAP the LIST function over the list of matches:

APPLY-FUNCTION
LIST `start-of-window n`
REPLACING `n` WITH `matches`

DEFINE number-pairs as this function (i.e., APPLY-FUNCTION …)

If all has gone well, you should have the number pairs necessary for PLOT to display one line of a dotplot of `seq1` vs `seq2`. To generalize, i.e. repeat these operations over all windows within `seq1`, we need a loop.

2i. Bring down a FOR-EACH loop. We'll treat `seq1`, `seq2`, and `window-size` as constants for the moment. The loop will presume they already exist. The loop will repeat lines **2e** through **2h** and loop through all values of `start-of-window`, FROM . Open up the body of the FOR-EACH loop and create holes for each function made in lines **2e** through **2h**. Then drag each of those four functions (in order) into one of the four holes.

2j. Open up a primary control *number from n1 to n2* box. Replace *var*iable with `start-of-window`. The first value should be 1 of course, so that the window begins at the beginning of `seq2`. What about the last value? Surely it's related to the length of `seq2`. Is it equal to the length? Devise a calculation for the last value and put it in the last value box. If you don't know how to calculate the number, make up a simpler case, say a sequence composed of just 11 letters.

2k. Open up a Results Section, using WHEN… APPEND. Why WHEN? You don't always want to add to the results, just when you found a match. Why APPEND? The difference between APPEND and COLLECT is subtle. We may explore it later. For now, trust me.

2l. WHEN what? WHEN matches has a value. That's expressed simply as WHEN matches. Put the variable matches in the condition box.

2m. APPEND what? APPEND all the number-pairs you found. Put the variable number-pairs in the value box.

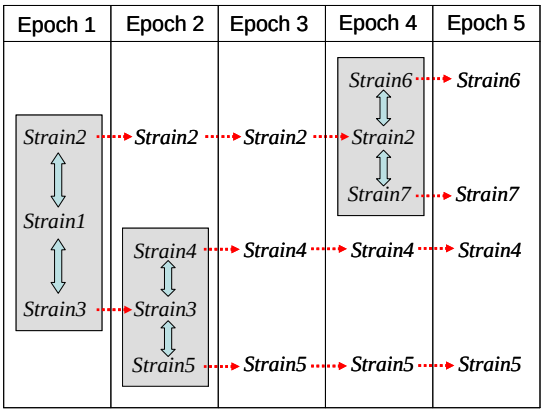
2n. Execute the loop. Do you get the expected collection of number pairs?

2o. Put the loop within a PLOT function. The easiest way to do this is to mouse over the action icon of FOR-EACH, click Surround, and then click PLOT in either the INPUT-OUTPUT menu or the ALL menu. The PLOT function by default plots lines, but you're doing a ***dot*** plot, so choose the POINTS option from the option menu of PLOT. Execute the resulting function.

2p. How do you interpret the dotplot?

3. Regardless of the focus of your research project, there is an excellent chance that the time will come when you wish to compare protein or DNA sequences from different phages and predict their evolutionary relationships. A family tree would be nice… how to do that?

But proteins, dotplots,… we've been fooling around long enough. The time has come to evolve ***Life***! What, after all, is life except replication with the possibility of change? If you agree, then here's the plan (see diagram to the right): We'll create a strain (call it *Strain1*), let it replicate and occasionally mutate, and see what we get in several epochs.



The fact is, however, we don't have the ability to watch events unfold over epochs. We can see only what's present today. So the next step will be to erase all knowledge of the course of the events we created and see if we can recreate those events solely on the basis of the final results.

3a. You'll be using the MUTATE function in this simulation of life. In PhAnToMe, bring it into your workspace by using RUN-FILE (**Input-Output** menu). Type in the *file-name* box `"mutate.bike"`, select the SHARED option, and execute the function. If all is well, you should now have the MUTATE function on your **Function** button.

3b. Test the MUTATE function by bringing it down from the **Function** button, entering `"AAAAA"` into the *sequence* box, and executing the completed function. Play with it a bit. Do you see what it does?

Epoch 1

3c. DEFINE `Strain1` as a random DNA sequence 40 nucleotides long using the RANDOM-DNA function (**Strings-Sequences** menu, **String-Production** submenu). Use the LABELED option of DEFINE, associating the name of the variable with the sequence.

3d. DEFINE `Strain2` as the SEQUENCE-OF *Strain1* (again LABELED), and DEFINE `Strain3` the same way. During Epoch 1, these strains are identical.

Epoch 2

3e. MUTATE `Strain2`, by putting `Strain2` in the *sequence* box, and executing the completed function. Do the same with `Strain3`, but don't mutate `Strain1`. Every epoch, mutations arise in the strains, but we'll leave `Strain1` frozen in time.

3f. Define `Strain4` and `Strain5` as the SEQUENCE-OF `Strain3` (again LABELED). During Epoch 2, these strains are identical.

Epoch 3

3g. MUTATE `Strain2`, `Strain4`, and `Strain5`. Leave `Strain3` frozen in time.

Epoch 4

3h. MUTATE `Strain2`, `Strain4`, and `Strain5`.

3i. Define `Strain6` and `Strain7` 1 as the SEQUENCE-OF `Strain2` (again LABELED). During Epoch 4, these strains are identical.

Epoch 5

3j. MUTATE `Strain4`, `Strain5`, `Strain6`, and `Strain7`. Leave `Strain2` frozen in time.

Present day

3k. We now have four evolved strains (Strains 4, 5, 6, and 7) and three strains frozen in an ancient time (Strains 1, 2, and 3). Compare their sequences by aligning them by means of the ALIGNMENT-OF function (available from the **Strings-Sequences** menu, **Bioinformatic-tools** submenu). Note that this function requires a single list of sequences, so you'll need to put a LIST function (available from the **List-Tables** menu) into the *sequence-list* argument of ALIGNMENT-OF, create seven boxes within LIST (through the **More** icon), and populate them with the seven sequences you've defined. Does the alignment make sense? Can you identify the mutations?

3l. Make a tree from the alignment by dragging the completed ALIGNMENT-OF function into the argument box of TREE-OF (available from the **Strings-Sequences** menu, **Phylogenetic-tree** submenu). Does the tree make sense, in light of how the strains evolved?

3m. Do the same thing, but this time with only the four present day strains (i.e., delete Strains 1, 2, and 3 from the list to be aligned).

From trees such as these, you can analyze present day sequences, inferring the events that led to them.

4. Now build a tree from real sequences. How closely related are different phage? That's a surprisingly difficult question to answer, owing to mosaicism (as discussed in the Hatfull review article). But we can readily assess the relatedness of specific phage ***proteins***.

4a. All bacteriophage should have tails. Find the tail proteins in phage Che12, using the GENES-DESCRIBED-BY function.

4b. Choose the two minor tail proteins. What phage proteins are they similar to? Use each of the two proteins as a query to SEQUENCE-SIMILAR-TO and `all-phage` as the target (using the IN option). Be sure you're doing a PROTEIN-VS-PROTEIN comparison. Are the two proteins similar to each other? Which protein has the most similar phage proteins?

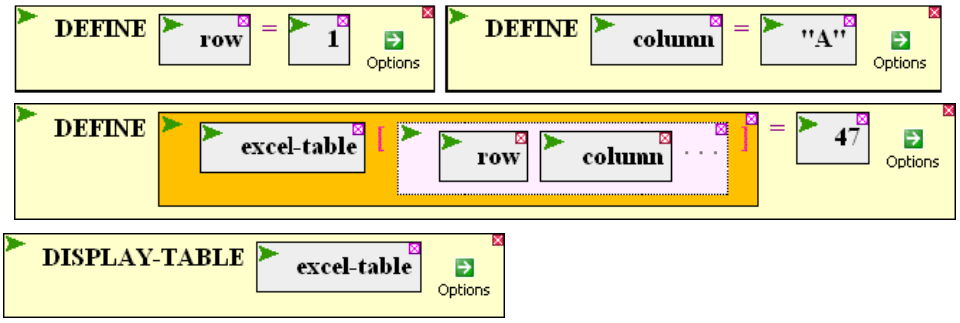
4c. Choose the minor tail protein with the most similar phage proteins and DEFINE a variable containing those similar proteins. Do this by dragging the completed SEQUENCE-SIMILAR-TO function to the *value* box and choose the RETURN-TARGETS option of SEQUENCE-SIMILAR-TO. This causes the proteins found by Blast to be returned by the function and assigned to the variable.

4d. Make a tree of those proteins, using the TREE-OF function on the ALIGNMENT-OF the variable you just created. What groupings do you discern?

Tables and the [ ] notation

It is often incredibly helpful to be able to create a set of information and refer to individual elements of the set. For example, you might consider the subjects of an experiment as S_1, S_2, S_3, etc. Or you might want to divide information into different categories, such as GNP_{Europe}, GNP_{China}, etc. In computer languages, this is achieved through data structures sometimes called arrays (if the subscripts are numbers) or hashes (if the subscripts are anything else). In BioBIKE, the two are combined into a concept called *Tables*.

5. You can create something like an Excel table by defining the contents of each cell in the following way:



Note the use of the **[ ]** function (obtainable from the **LISTS-TABLE** menu) to access a specific cell of a table. Replace `row` and `column` with different numbers and letters and re-execute the definition. Then display the table. From these experiments, how does BioBIKE think of tables?

6. From the **Variables** menu, **Expunge** the `excel-table` variable you created in Question **5** (so you can start fresh). Then put the definition of `excel-table` into a loop in such a way that you define a table with rows from 1 to 10 and columns from "A" to "J" (which may be accomplished by creating a loop variable that is IN a set defined by FROM "A" TO "J"). The content of each cell should be the row JOINed with the column. For example, the content of cell 1A would be "1A". Make and display this table.

7. Make and display a table of squares, where the index is a number from 1 to 100 and the value of the table is that number squared.

8. Make and display a table containing information about the cyanobacteria known to BioBIKE. If you're in PhAnToMe, you can get a list of all cyanobacteria using the **Bacteria (phylo)** menu from the **Organisms** button. The row labels should be the short names of the organisms (using the NAME-OF function with the SHORT option). The column labels should "size", "gene-number", and "GC-fraction". If you're irritated by quotes, you can use as labels 'size, 'gene-number, and 'GC-fraction.