Bioinformatics M.Sc Module: BIOL60201

Project 1: A Python Proteome Analysis Pipeline

This project is designed to be completed by a group of 4 people, but 3 should still be able to manage it comfortably with some help. The project is divided into a series of Tasks, each of which will require some development of a simple Python program or will require some execution of the code supplied for groups smaller than 4. The Tasks will lead in to, or follow on from, another task, and hence it is essential that you think carefully about the formats your programs will be able to cope with, and that you draw up sensible specifications *before* you all start coding.

Overview

You will be given a genome sequence, from a bacterial organism, which has just been sequenced. An experimental group working on this organism wishes to carry out some proteomics experiments to identify proteins from the microbe. However, they need a set of proteins in a FASTA formatted database in order to search against with their mass spectra. They then want to know which is the best digestive enzyme to use, so that they get the best chance of identifying as many proteins as possible. In other words, they want to find the enzyme which produces the lowest number (on average) of peptides per protein. This is because this will be more diagnostic, since the peptides are rarer. However, their mass spectrometer can only measure peptides which have a mass (or strictly speaking, mass-to-charge ratio) between 1000 and 1500 Da and it has a maximum mass accuracy of 0.2 Da.

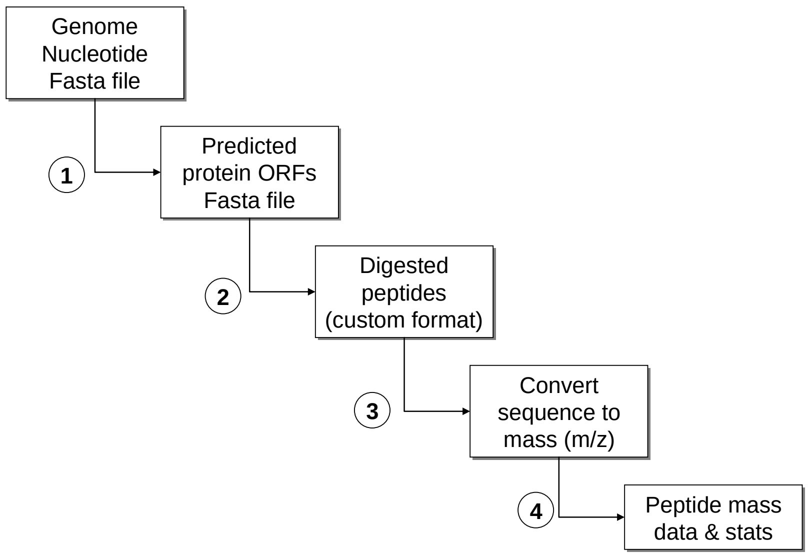
Genome Nucleotide Fasta file
1
Predicted protein ORFs Fasta file
2
Digested peptides (custom format)
3
Convert sequence to mass (m/z)
4
Peptide mass data & stats

Figure 1. Flow chart of proteome project

The tasks are described in more detail below.

Task 1 – An ORF finder

You will need to write a Python program which can read in a Fasta format file containing nucleotide sequence (DNA sequence, containing ACGT) and find all the possible open reading frames (known as ORFs, beginning with ATG[1], and ending with a stop codon, TGA, TAA, TAG). You can also assume that

[1] Note: >90% of codons in bacterial genes start with ATG but around 10% don't. However, we'll assume they all do for the purposes of this exercise.

the longest ORF will be used, and not have to worry about any internal ones. You will have to worry about all 6 open reading frames though, and predict ORFs in all 6.

Your program should accept standard Fasta format as input, of the form:

```
>Genome sequence
AATTCAGTTACTTATTCCCCTTATTGGCAGTATTAACGCATAGGACTGCG
ATAGACTTTACTAAGCTCAGTATCATTCCCTTATTGGCATAGTTACGCAT
CCACCAATAGACTTTACTATTCCCTTCATGGCATATTTACGCAGTAGGAC
ATAGACT
```

And should output protein ORFs with the following fasta format:

```
>orf_name frame length start
MAKSKDFPVKADFAAHVAIQSEFAHGV
>orf_name frame length start
MHILDECCAKSKDFPDFAAHVAYIQSEFAA
LLILPQWNSDAAHGV
```

Orf_name should be a unique name for the ORF, which might include the name of the genome/organism, frame in which the ORF is found, and some number (ie. if the organism is called *I.claudius* you could call your ORFs something CLAUD_F1_0017 (for the 17th ORF in frame 1 in *I.claudius*).

Output should be written to standard output.

Please take note: this genome, like most, contains a few bases which were not called unambiguously. This means that they are given the letter ‘n’ instead of A, C, G or T. Obviously, these can not be formally translated in an amino acid in most cases. I suggest you ignore all ORFs that contain a codon with a ‘N’ in it.

You should also think carefully about the minimum size of an ORF. I suggest you add an option which controls this, and I would use 50 amino acids as the minimum size, or perhaps even 100 amino acids (= 300 nucleotides).

Bonus tasks:

Deal with more than one sequence, and more than one file supplied on the command line as arguments

Add some command line options in, such as the ability to write to a specified output file instead of standard out with *–o filename*, or restriction to just one frame with *–f frame_number*.

Provide a short “usage” output if the user puts a wrong option or supplies incorrect arguments, or perhaps uses the –q option.

Error checking: spot improper formats, unknown amino acids, and correctly deal with “gaps” (there normally won’t be any of course).

Only taking longest ORF when the ones in different frames overlap – this is actually very tricky.

Task 2 – A protein digester

This python script should read in the output from task 1. ie. it should be able to cope with one or more protein sequences, and then digest them into smaller peptide sequences, reporting these back one per line. The script should be able to cope with a user-defined choice of digesting enzyme, from the following set:

Trypsin: cuts at Lysine (Lys, K) or Arginine (Arg, R) unless the next amino acid is Proline (Pro, P).

Endoproteinase Lys-C: cuts at Lysine (Lys, K) unless the next amino acid is Proline (Pro, P).

Endoproteinase Arg-C: cuts at Arginine (Arg, R) unless the next amino acid is Proline (Pro, P).

V8 proteinase (Glu-C): cuts at Glutamic acid (Glu, E) unless the next amino acid is Proline (Pro, P)

The program should read in Fasta formatted protein sequence e.g.:

```
>protein1
ALTAMCMNVWEITYHKGSDVNRRASFAQPPPQPPPPLLAIKPASDASD
```

And report the individual peptides after digestion (eg. For trypsin)

```
>protein1 peptide 1
ALTAMCMNVWEITYHK
>protein1 peptide 2
GSDVNR
>protein1 peptide 3
R
>protein1 peptide 4
ASFAQPPPQPPPPLLAIKPASDASD
```

Bonus tasks

Adapt your code so the program will also consider missed cleavages. These are instances when the enzyme misses out on a putative cutsite. You can allow up to 1 missed cleavage, for example, so that

```
AHLKNMILQEWRMAALI
```

Would produce `AHLKNMILQEWR` and `NMILQEWRMAALI` on top of the standard peptides `AHLK NMILQEWR MAALI` (ie. its inclusive, so that you calculate up to 1 missed cleavage). This should be supported by a command line option to allow the user to select the number of missed cleavages.

Add some error trapping so the program spots "unusual" (B,O,U,J,Z) or unknown amino acids (X)

Task 3 – A mass analyser

In this task you should read in a standard Fasta format file and convert the sequence information into masses, or strictly speaking, mass-to-charge values. You will find the masses of the standard amino acids to 4 decimal places here (http://www.matrixscience.com/help/aa_help.html). The program should cope with both monoisotopic and average masses, and should calculate the mass-to-charge value as if they were all ions of +1 charge (Hence, you need to add 1 to all values, for 1 proton). You also need to add the mass of water to the mass of the summed amino acid values, as these are for a single residue, so you need to add the masses of the terminating groups (e.g. H at the N-terminus and OH at the C-terminus). These values are: 18.0106 for monoisotopic masses, and 18.0153 for average masses. Your output should be something of the form.

```
Prot_name      peptide mass-to-charge  z     p     sequence
ABC1_YEAST        1    1234.5678       1     0     ALTAMCMNVWEITYHK
ABC1_YEAST        2    894.3019        1     0     SSLPMAR
ABC1_YEAST        3    1031.6791       1     0     MNASFFAAQL
```

Where "peptide" is the number of the peptide in the protein, z is the charge of the ion (+1 by default), and p is the number of missed cleavages (you could calculate this yourself, or perhaps get Task 2 to pass this information on ?).

All of the options should be provided on the command line.

Bonus tasks
Consider some standard post-translational modifications, such as blocking of reduced cysteines, oxidation of methionines and phosphorylation of Ser, Tyr and Thr.

The script should be able to report only N-terminal peptides (ie the first one in a protein) or just the C-terminal ones (the last one in a protein), selectable from the command line as an option.

Task 4 – the ion statistics calculator

This task should read in the output from task 3 and calculate some statistics and export the results, to the screen or a file. It should be able to report the number of peptides in a given mass range. Ie. calculate the average number of m/z values found within a m/z range (or set of m/z bins) specified by the user on the command line. For the binned histogram data, the bin sizes should be user-definable and would allow you to model the mass accuracy of the instrument. It should also be able to selectively count ions if they match a certain sequence pattern (ie. only ions which contain a Cys, or ions which start with a Met). The utility of this simple script will be self-evident when you read what you have to do in the next section. The whole team should be involved to help design the question this tool needs to be able to answer. So think carefully (as a team) as to what makes an enzyme more (or less) useful for proteomics? Although you may end up plotting the data yourself with other tools, or possibly using Python Notebooks, the script itself should just calculate the basic numbers and export CSV or tab-delimited text (not plots or pictures) which could be used by other tools.

Bonus tasks
Adapt the script so that it can cope with mass accuracy supplied as parts per million (ppm) instead of Da.

Overall aims

The overall aims of this project are to use your scripts to calculate the most effective identification system for a proteomics experiment under given (user-specified) conditions. A more effective system will be obtained if the proteomics researchers have a smaller number of peptides in the given mass (or more strictly, m/z) range, as any peptides they do find are more likely to be completely diagnostic. They might even be totally unambiguous – ie. there is only one protein which has a certain peptide with a certain m/z under certain conditions (Enzyme = Arg-c, missed cleavages = 1, N-terminal amino acid = Ser etc). You should conduct some focussed investigations with your programs, to answer the specific question:

Which is the best enzyme to use considering singly charged ions (+1), and m/z ratio between 1000 and 1500 Da, and up to 1 missed cleavage (ignore all post-translational modifications, additional sequence information etc.).

As well as more general investigations to see how the number of peptides per protein is affected by changing the full range of options.

Data

The genome sequence you need to work with will be uploaded on to the SageMathCloud environment, as well as the Blackboard site, in the file `genome.fasta`

Write-up

The write-up for this project is intended to short, with two components – a group report on the pipeline as a whole (see below) and the individual programs alongside a man page description of what they are and how to use them. This should therefore be a single page of description for each script, which will be a Unix "Man page" style description of the operation of the script, its aims, options and results. This should be followed by the actual script itself (in a fixed width font please, which can be quite small if you like). The script should have lots of comments to aid readability. Each person should prepare the "Man page" for their code, and the authors should be clearly marked.

You should also prepare (as a group) a short report (3 pages max) on which is the most effective enzyme, and why, as well as any general observations about what makes proteome protein identification easier (or harder) based on your observations.

Please submit all work via **Blackboard.** This should include all the individual reports (man page + script), as well as a single report per group. There should be two assignments on Blackboard to support this, one for each of you to submit individual work, and one to submit the group report.

Assessment

The marks for this work will be awarded as follows:

40 marks for each script awarded to the author – this **must** be **<u>your own work</u>**.
10 marks as a group mark for the report and general performance of the scripts.

Each person has a mark out of 50 by summing the individual and group marks

Deadline: 4:00 pm Thursday Nov 15th