[PER] EVALUATION PROJECTS

by Josep F. Abril

<> BACK <>

Overview

As it was agreed during this subject, we were going to evaluate the programming skills achieved with a small project. This page describes several projects, from which **you should take one**. If you dare, you can even work on and submit two. The programming projects are sorted by difficulty, from easy to hard ones (not so hard, really). Accordingly, each project section has a table showing the maximum punctuation a student can obtain at three different levels: basic implementation, modularity of the code, and reporting (both implementation issues and results). However, it is *conditio sine qua non* that the scripts submitted have to work without errors/warnings to achieve the corresponding grade, **otherwise they will not be considered**. All the scripts must contain the `strict` and `warnings` pragmas.

Submission Instructions

IMPORTANT !!! Bear in mind the following instructions before submitting your programming project...

You have to submit a `zip` or `tgz` file having a main folder (the folder name should include the submitter's name and surname). That folder shall contain the **Perl script** required to solve the proposed problem and a short report, in **PDF format** (when working with Office-like software, there are many tools to export to PDF), discussing the results obtained. If you like, you can include the data obtained in the analysis, although it is better not to send that by email. Data plots can be made with the R package or using Excel-like applications; in any case, figures must be included in the PDF report.

All project submissions should be made through the AulaGlobal-Moodle web site for the [PER] subject. Attendees not registered can consider also submitting a project, but they have to send the packed file by email. Submission deadlines were already stated at class. **No submissions would be considered after that deadline**, therefore they will not be graded.

Contents

- CALCULATING SOME STATS FOR A SET OF NUMBERS
  - Calculate Stats for a Given Field from Tabular Files
  - Calculate Stats for a Set of Fields from Tabular Files
- ANALYZING COMPOSITION of GENOMIC SEQUENCES
  - GC-Content of Genomic Sequences
  - Running Windows of GC-Content
  - Estimating Stop Codons Frequencies
  - Estimating Hexamer Frequencies
- RECORD FORMAT CONVERSION
  - Converting FASTQ sequences into FASTA and QUAL files
  - Merging FASTA and QUAL files into FASTQ
- WORKING with GFF RECORDS
  - Extracting GFF Features from Sequences
  - Filter longest transcripts for a set of gene annotations in GFF
  - Intersecting GFF Features from Two Sets
- MORE GENOMIC ANALYSES with PERL
  - Mapping RNA Secondary Structure into an Alignment
  - Extracting Intronic Features from Exonerate Output
  - Getting the Topmost Scoring Sequences from PWMs

CALCULATING SOME STATS FOR A SET OF NUMBERS

Calculate Stats for a Given Field from Tabular Files

One can easily reuse the Perl script that computes the mean value of a set of numbers from the examples section of the *Scalar Vars* lecture. The script should read a given numerical field from a file in tabular format (like this file: numbers.tbl), and load numbers into an array variable. This means it has to take into account two arguments from the command-line, the column and the input file. Make this script to return several statistic measures for the same set of numbers: the sample size, the minimum, the maximum, the 1st, 2nd and 3rd quartiles (25%, median and 75% respectively), the mean, the variance and standard error.

```
sh$ ./getstats.pl 1 numbers.tbl
Stats for column: 1
  Cnt: NNN
  Min: XXX.XX
  1qt: XXX.XX
  Med: XXX.XX
  3qt: XXX.XX
  Max: XXX.XX
  Avg: XXX.XX
  Var: XXX.XX
  Std: XXX.XX
```

GRADES		
	Basic implementation (*working bug-free script*)	**5.0** pt
	Using functions	**+ 0.5** pt
	Report describing implementation and results	**+ 0.5** pt

Calculate Stats for a Set of Fields from Tabular Files

An extension from previous script could be a program to calculate all the requested stats not only for a single column, but for a set of columns. Using a character as a field id separator, like ":", split down the first argument to define an array of columns for which we want to compute the corresponding stats.

```
sh$ ./getstats.pl "1:4:7" numbers.tbl
Stats for column: 1
  Cnt: NNN
  Min: XXX.XX
  1qt: XXX.XX
  Med: XXX.XX
  3qt: XXX.XX
  Max: XXX.XX
  Avg: XXX.XX
  Var: XXX.XX
  Std: XXX.XX

Stats for column: 4
  Cnt: NNN
  Min: XXX.XX
  ....
```

GRADES		
	Basic implementation (*working bug-free script*)	**5.5** pt
	Using functions	**+ 0.5** pt
	Report describing implementation and results	**+ 0.5** pt

ANALYZING COMPOSITION of GENOMIC SEQUENCES

GC-Content of Genomic Sequences

Given two sequences of two different organisms, we need to determine to which one a third sequence belongs to. There are different measures we can use to achieve this, calculate the GC content is the simpler one. You can adapt the Perl script from this section of the *Scalar Vars* lecture. Make it able to read a fasta record from a file instead of pasing the sequence from a variable. You can also adapt the script from this section of the *Reusing Code* lecture. Then, you will compute easily the GC content of all the sequences from the files provided as in the following example:

```
sh$ ./getGCcontent.pl drome_seq.fa
GC-content: XX.XXX%

...$
sh$ ./getGCcontent.pl query1seq.fa
GC-content: YY.YYY%
```

```
...$
```

You can download the genomic sequences from the following links:

[Fruitfly chr3R:9240000-9264999] [Mosquito chr2L:9395001-9420000]

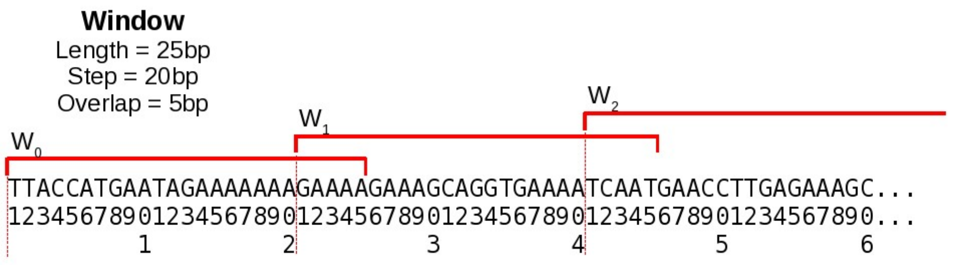
[QUERY 1] [QUERY 2] [QUERY 3]

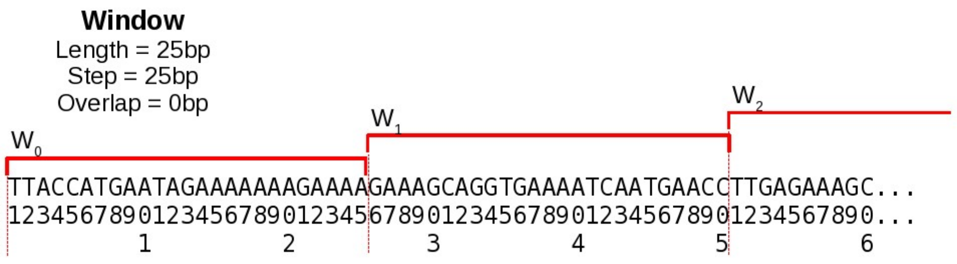
You have to hypotesize for each of the three queries, which of the two species is more likely to belong to, based on their GC content. Discuss briefly your results.

GRADES		
	Basic implementation (*working bug-free script*)	**5.0** pt
	Using functions	**+ 0.5** pt
	Report describing implementation and results	**+ 1.0** pt

Running Windows of GC-Content

We want to visualize the GC content of running windows along a genomic DNA sequence. We will be able to observe if the GC content levels are more or less constants along the sequence. Otherwise, we will localize those regions having the most dramatic changes in GC content, which can be due to different factors, i.e. a greater gene content. Adapt the script from this section of the *Reusing Code* lecture (you can also take the function-based implementation on the same lecture as template, if you wish). You have to include a loop in order to scan this genomic sequence of the fruitfly at chr3R:9240000-9264999, with a running window that has to be defined as a parameter of the script command-line. The script has to print out a table with the middle nucleotide position of each window and the GC-content of the nucleotide substring defined by the corresponding window. For those cases in which one has to work with "N"-masked sequences, it is advisable to apply a correction factor to the GC content. When the length of the masked substring is larger than half of the length of the runnign window, then it is better to return a "NA" score (from "Not Available"). The following schemma illustrates some of the concepts used when talking about the running windows:





The second parameter for the script has to be the window step; the third the name of the file that contains the fasta-formatted sequence; as shown below:

```
sh$ ./windowGCcontent.pl 1000 500 drome_seq.fa
500  XX.XXX%
1000 XX.XXX%
1500 XX.XXX%
2000 XX.XXX%
...

sh$
```

You have to make a plot of the nucleotide position along the sequence (X-axes) versus the GC content (Y-axes), and discuss the results obtained with the script. You can deepen into this analysis by calculating the scores at several windows (i.e. 100bp, 500bp, 1Kbp, 2Kbp and 5Kbp). In that case, you can also suggest which window will yield the best results when extrapolated to the whole genome.

GRADES		
	Basic implementation (*working bug-free script*)	**5.5** pt
	Using functions	**+ 0.5** pt
	Report describing implementation and results	**+ 1.0** pt

Estimating Stop Codons Frequencies

Write a script that reads a sequence in fasta format from a file and scans it nucleotide by nucleotide to account for all occurrences of each of the three stop codons (say here TGA, TAA and TAG). You have to scan sequence on both strands, forward and reverse, so that you have to read nucleotides from both ends. Remember that you have to obtain the reverse complement of each tuple of three nucleotides, when scanning from the end of the sequence towards its start, in order to find the stop codons of the reverse strand. Finally, the script should write the relative proportion of each stop codon with respect the total codons that can be obtained from that sequence and with respect the other stop codons. You can adapt the script from this section of the *Reusing Code* lecture, for instance. Here, you have an example of the command-line and the expected output:

```
sh$ ./stopCodonFreqs.pl drome_seq.fa
# CODON RelFreqAll RelFreqStp
# TGA   xx.xxx%    xx.xxx%
# TAG   xx.xxx%    xx.xxx%
# TAA   xx.xxx%    xx.xxx%
# ORF LENGTHS
10
200
...

sh$
```

Compare the results for two genomic sequences from two different species, Fruitfly chr3R:9240000-9264999 and Mosquito chr2L:9395001-9420000. Can you determine if there is a stop codon that appears more frequently than the other ? Do we observe differences between the two species or they have similar frequency distributions ?

To make the script even more useful, you can also account for the length of all the open reading frames (ORFs), those genomic regions that are surrounded by two stop codons. In this case, you have to plot the distribution of ORF lengths for both species sequences too.

GRADES		
	Basic implementation (*working bug-free script*)	**6.0** pt
	Using functions	**+ 0.5** pt
	Report describing implementation and results	**+ 1.0** pt

Estimating Hexamer Frequencies

It has already been described that coding sequences have different intrinsic properties than those of intronic or intergenic. One of those distinct features is the modulus 3 periodicity due to the accumulation of G and C nucleotides on the third codon position, mainly explained by the redundancy of the genetic code allowing silent mutations on that third position. A part from that, protein structure will introduce further constrains on the amino acid composition, which will turn also into constrains at nucleotide level composition. From two sets of sequences, one set filtered out from real coding-exon sequences and another from intronic ones, we need a script that will calculate frequencies of hexamers for the two sets. Then, given a third file with some unannotated sequences, the program should predict which ones are coding and which are intronic. You can discuss on the report if there is any other factor, such as sequence length or biases on the training sets, that may have an impact on this prediction.

We can compute the log-likelihood ratio of finding a given hexamer as the logarithm of the ratio of the observed frequency for that hexamer in coding-exons, divided by its frequency under the background model. This background model can be a random model, but we hypothesize it can increase the signal-noise ratio if we use the observed frequencies of hexamers from introns instead. Assuming that there are no other dependencies than those from the di-codon (or hexamer), thus assuming independence between consecutive hexamers; we can calculate the probability of being coding as the product of the frequencies each hexamer from the query sequence (which translates into a sum in logarithmic scale). For a given hexamer sequence $S = H_1, H_2, \dots, H_m$,, we define hexamer score as:

$$\text{Hexamer Score} = \frac{1}{m}\sum_{i=1}^{m}\log\left(\frac{F(H_i)}{F'(H_i)}\right)$$

Therefore, hexamer score will determine the relative degree of hexamer usage bias in a particular sequence: positive values will indicate a coding sequence, whereas negative values will indicate a non-coding sequence.

Just three final remarks:

- first, take the base 2 logarithms to get scores in bits scale;
- second, you must consider the correct in-frame hexamer frequencies from the training set (look for the exon frame to) and then check the hexamers from the different frames of the input sequence, in order to find the frame having the best score;
- third, in-frame hexamers can be obtained with a 6 nucleotides running window, considering a 3 nucleotides step (take a look to the sliding window figure from previous GC-content section).

Here you have the links to three FASTA files (compressed using `gzip` command): coding.fasta.gz (2000 seqs), intronic.fasta.gz (2000 seqs), and unknown.fasta.gz (20 seqs).

GRADES	Basic implementation (*working bug-free script*)	**7.0** pt
	Using functions	**+ 1.0** pt
	Report describing implementation and results	**+ 1.0** pt

RECORD FORMAT CONVERSION

Converting FASTQ sequences into FASTA and QUAL files

FASTQ format is a text-based format for storing both a nucleotide sequence and its corresponding quality scores for each single base. You can find further information about this sequence format at this Wikipedia link. In a FASTQ file, the quality scores are encoded into single characters, given a specific mapping over a set of ASCII symbols (see here different encoding methods). Those scores are integer values that correspond to the base call accuracy in terms of the decreasing probability of an incorrect nucleotide assigned to a sequence (further info on Phred Quality Scores). Take into account that FASTQ records have a fixed number of lines per sequence, 4 in this case, while FASTA and QUAL multi-line records do not have a priori fixed number of lines to store the nucleotide sequence or the scores, respectively.

Although novel Bioinformatics tools may have support for FASTQ, we may have to deal with many others that will require that sequences were encoded in the old standard FASTA format. Thus, the main goal of this project is to implement a filter that will extract from a FASTQ file, the nucleotide sequence and the scores into two separate output files, both in FASTA format (see the the examples below to understand the relationship between FASTQ, FASTA and QUAL record formats).

```
## EXAMPLE sequence/score in FASTQ format
@EXAMPLE 94bp
ACAGGAGACAGAGAGACATGCAGAGAGACAGGAGGCAGAGAGACAGGACAGGAGACAGAGAGACATGCAGAGAGACAGG
+
FFFFFFFFFFFIIIIIIIIIIIIIIIIIIIIIIBBBHHHHHCCAAA=FFFFFFFFFFFIIIIIIIIIIIIIIIIIIIII

## Above FASTQ nucleotide string in FASTA format
>EXAMPLE 94bp
ACAGGAGACAGAGAGACATGCAGAGAGACAGGAGGCAGAGAGACAGG
ACAGGAGACAGAGAGACATGCAGAGAGACAGG

## Above FASTQ score string in QUAL format
>EXAMPLE 94bp
37 37 37 37 37 37 37 37 37 37 37 40 40 40 40 40 40 40 40 40 40 40 40 40 40
40 40 40 40 40 40 40 40 33 33 33 39 39 39 39 39 34 34 32 32 32 28 37 37 37
37 37 37 37 37 37 37 37 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
40 40 40 40
```

You can download a small FASTQ file that has to be converted to FASTA and QUAL from this link: reads.fastq.

GRADES	Basic implementation (*working bug-free script*)	**6.0** pt
	Using functions	**+ 0.5** pt
	Report describing implementation and results	**+ 0.5** pt

Merging FASTA and QUAL files into FASTQ

It may happen, when analyzing NGS (Next-Generation-Sequencing) data, that you need to include older sequence sets in FASTA format and that the software you were using only can deal with FASTQ formated sequences. You can take a look to the links on the previous subsection to get more information about those file formats, as well as examples of their record structure from that subsection gray box.

Thus, here we have to build a filtering script that can merge two synchronized files in FASTA format into a single output file in FASTQ. By synchronized we mean that both files, FASTA and QUAL, must have the nucleotide sequences and the related scores in the same order for all the sequences stored on them. Another problem is to convert a FASTA file for which we do not have the corresponding QUAL file. In that case, our script must take care of how many input files were provided on the arguments from command-line, so that in case no QUAL filename is given after the FASTA filename then it will create a fake scoring string for the output FASTQ records.

You can download two short example files that have to be merged into a FASTQ, from this link: reads2.fasta and reads2.qual. You can analyze sequence content and scores distribution for each processed sequence and the average values for all the sequences; then output that result to the standard error I/O channel. There is an important issue to take into account, what happens when the FASTA and QUAL files contain the same sequences, but they appear in different order. For this purpose, you have a second set of FASTA and QUAL files, that you have to convert into FASTQ too: reads3.fasta and reads3.qual.

GRADES		
	Basic implementation (*working bug-free script*)	**7.0** pt
	Using functions	**+ 0.5** pt
	Report describing implementation and results	**+ 0.5** pt

WORKING with GFF RECORDS

Extracting GFF Features from Sequences

Here, we want to retrieve those substrings from a DNA sequence that are delimited by the start and end coordinates of a GFF record. It is straighforward to adapt the script from this section of the *Perl Input/Output* lecture, in order to read from a GFF file instead. Then, add a while-loop to read a sequence in fasta format from another file, before parsing the GFF one. Finally, iterate through each start-end coords pair from the GFF records and use the `substr` function appropriately, in order to extract the corresponding sequence and printing to standard output along with the coordinates and the feature identifier. An example of its command-line is shown below:

```
sh$ ./getseqfeatures.pl drome_seq.fa drome_genes.gff
exon1.group 100 150 CGTTGGAT...ATTGAT
exon2.group 340 410 ATTGGATG......CTGTTG
...

sh$
```

We provide a genomic sequence of the fruitfly at chr3R:9240000-9264999, along with two RefSeq genes that are annotated in that genomic region. Just a couple of remarks: start and stop coordinates correspond to the fourth and fifth columns of every GFF record (feature and group are the third and tenth, respectively); take into account that sequences for GFF features on the reverse strand must be reverse-complemented; remember that Perl strings start at index 0, then apply this coords correction *start - 1*; if you keep record of the previous coords fix, then you can get the length of the substring by simply substracting that number to the stop coords. Once sequences have been extracted, you can check if the GC content for the CDS features is different from that of the introns (you can obtain intron coords from two consequtive CDS features of the same gene).

GRADES		
	Basic implementation (*working bug-free script*)	**6.0** pt
	Using functions	**+ 0.5** pt
	Report describing implementation and results	**+ 1.0** pt

Filter longest transcripts for a set of gene annotations in GFF

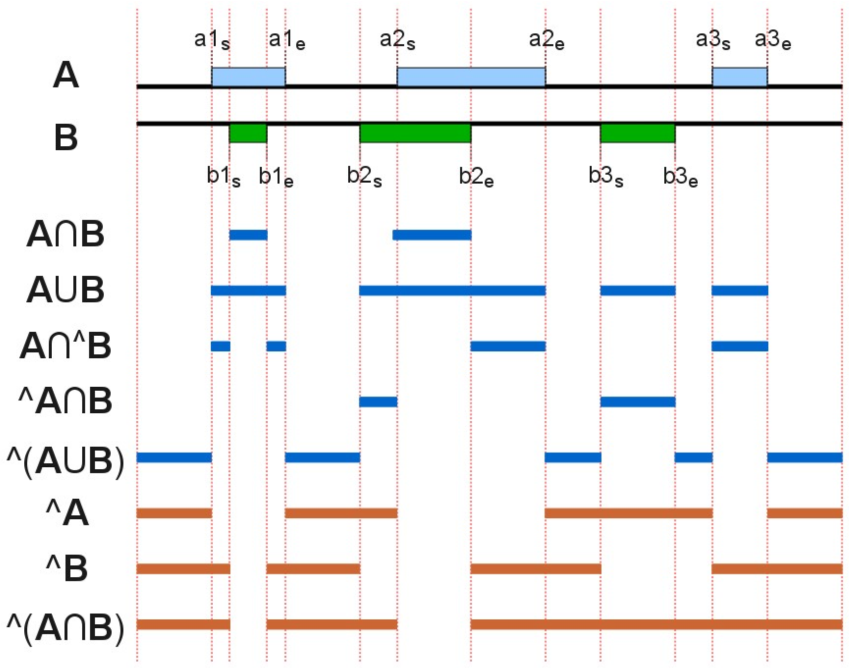
Due to the alternative splicing process a gene loci can contain one or more transcript isoforms. Given a GFF file where the coordinates of many genic structures for different genes were encoded, write a script that can filter out the features for the longest transcript on every gene of the file. You can choose to add a command-line option, so that the program can filter out the longest or the shortest. In order to achieve this from a GFF set of features corresponding to every transcript for every gene, you need to store somehow the smallest start and the largest end positions for those transcripts (if you consider the length of the transcript at genomic level), or the sum of the lengths of all the exons/CDS (if you consider the real length of the final messenger RNA produced by that isoform). After calculating the overall lengths, it should be easy to get the first or last element of the sorted-by-length list, and report to the standard output the feature annotations for such transcript on every gene. There can be genes having only one transcript, do you think this can affect the process ?

How genes and transcripts are annotated using GFF format may vary depending on the format version. However, the gene/transcript identifiers are often used as grouping features, and they are placed at the grouping columns (those from the 9th field and beyond). You have to be aware that the same gene name can appear on more than one sequence (specially when working over gene predictions). Here you have a file to be processed: genes.gff. Once you have filtered down the longest and shortest transcripts, you can analyse the size variation by gene and plot it (for instance, the distribution of the long-short difference).

GRADES	Basic implementation (*working bug-free script*)	**6.5** pt
	Using functions	**+ 1.0** pt
	Report describing implementation and results	**+ 1.0** pt

Intersecting GFF Features from Two Sets

Given the two GFF files provided (available from those links: file_A.gff and file_B.gff), make a script that will read them into two data structures, from now on named as **A** and **B**. Then apply some set operations based on the starting/ending coordinates and dump the corresponding coordinate projections. The set operations are: the **union**, return the contiguous projection of all overlapping features found in A, in B or in both sets; the **intersection**, return the projection of only those overlapping regions that are in common between A and B; the **A:B-complement**, those coords found in B that are not present in A; the **B:A-complement**, just the opposite as before; and the **complement** of both, those contiguous regions which are outside the coordinates provided in both sets. The figure below illustrates those sets; we are mainly interested on the ones described here (in blue), but you can also produce the three remaining ones (in brown). Take into account that GFF features are strand-dependant, so that you may need to project on both strands, forward and reverse, separately.

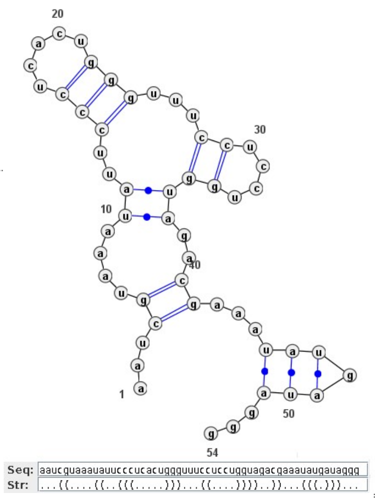


GRADES	Basic implementation (*working bug-free script*)	**7.0** pt
	Using functions	**+ 1.0** pt
	Report describing implementation and results	**+ 1.0** pt

MORE GENOMIC ANALYSES with PERL

Mapping RNA Secondary Structure into an Alignment

It's usual to encode secondary or tertiary structures for sequences as strings of specific sets of symbols. In this case, we have a string containing a set of nested parentheses, **SQstr**, which define sequence complementarity between the nucleotide sequences conforming the hairpins of an RNA molecule, namely **SQref**. We were provided with a set of similar RNA molecules that were aligned against our reference sequence. However, most RNA structure visalization tools cannot map the contacts string over different sequences. Thus, we need a Perl script that projects the contacts string for **SQref** on every sequence aligned to it. You must take into account that nucleotide substitutions can affect the pairings described by the set of parentheses. Furthermore, gaps in the alignment can delete sequence that is at the same position of an opening or closing paranthesis. So that, the script has to produce a specific contacts string for each sequence in the alignment but it has to ensure that all the parentheses are properly balanced. Those mismatched parenthesis should be replaced by a dot (".") in order to get a contacts string suitable for most visualization tools. The figure on the right shows the RNA secondary structure for **SQref**, it was generated by VARNA 2D viewer demo web server. You can include on the report a table with the adjusted 2D figure for each sequence.



Seq: aaucguaaauauucccucacuggguuuccucccugguagacgaaauaugauaggg
Str: ...((....((..(((.....)))...((....))))..))...(((.)))...

```
SQref aaucguaaauauucccucacuggguuuccucccugguagacgaaauaugauaggg
SQstr ...((....((..(((.....)))...((....))))..))...(((.)))...
SQv01 aaucguaaauaaucccucacuggguaaccucccugguagacgaaauaugauaggg
SQv02 -----uaaauauucccucacuggguuuccucccugguagacgaaauaugauaggg
SQv03 aaucguaaauauucccucacuggguuuccucccugguagacgaaauaug------
SQv04 aaucguaaauauucccucac-----uuccucc---uagacgaaauaugauaggg
SQv05 aaucguaa--aaucccucucuggguaaccucccuggu---cgaaauaugauaggg
SQv06 aaugguaaauauucccucucugggguguccucccugguagacguuauaugauaggg
SQv07 aaucguaaauauucccucacuggguuuccucccugguagacgaaauaugauaccg
SQv08 aaucguaaau----ccucacuggg---ccucccugguagacgaaauaggauaggg
SQv09 -------------cccucacugg----ccucccugg-----------gauaggg
SQv10 aaucguaaauauucccucucuggguuuccucccugguagacgaaauaugauaggg
```

GRADES		
	Basic implementation (*working bug-free script*)	**7.0** pt
	Using functions	**+ 1.0** pt
	Report describing implementation and results	**+ 1.0** pt

Extracting Intronic Features from Exonerate Output

Two ESTs were mapped into two genomic contigs using `exonerate`. The output from this program, including the pair-wise alignments and features in vulgar and GFF formats, was stored into two files: Contig73.exonerate.out and Contig130.exonerate.out. The genomic sequences for the genomic contigs are also available from this file: genomic_contigs.fa. We are interested in the analysis of the introns found and the corresponding splicing signals. The problem is that `exonerate` just clips the introns in order to produce the alignment. In order to study the intronic sequences we need to combine the sequences and `exonerate` output in order to produce the following datasets: 1) a tabular file for introns with two fields, the intron ID and its sequence; 2) a tabular file for the splice sites, having three fields, the intron ID, the donor and the acceptor sequences; 3) a file containing all the alignments from `exonerate` output, but clipped 10bp around the reported splice sites. The intron ID can be easily built recycling the numbering provided by `exonerate`, it should has an structure like *target_sequence_id*.INT.*number*, where *target_sequence_id* is the corresponding genomic contig ID and *number* the given intron number. The "intronic-neighbourhood" alignments requested at point 3, should contain the adjusted coordinates from query and target sequences. An example of such alignment is shown below:

```
QRY: SQXXXXXX     967 : CTATTCCGTC  <<<< Target Intron 4 <<<<  TTTCCGTANA :  986
                        ||||||||||++          44 bp          ++|||||||| |
TGT: ContigYYYY  4400 : CTATTCCGTCgt.........................agTTTCCGTAGA : 4463
```

GRADES		
	Basic implementation (*working bug-free script*)	**7.0** pt
	Using functions	**+ 1.0** pt
	Report describing implementation and results	**+ 1.0** pt

Getting the Topmost Scoring Sequences from Position Weight Matrices

Position Weight Matrices are a simple way to model signals appearing on DNA and protein sequences. They summaryze the frequencies of every letter of the nucleotide or amino acid alphabets at a given position of the signal, for instance a splice site or a transcription factor binding site. This means that we can obtain a score, either from the absolute frequencies or by transforming them into log-likelihoods, for a given sequence to determine if it contains the signal pattern or not. Yet, they can also be used as sequence generators. The idea is to write a Perl script to produce all the possible sequences a PWM can match and report them ranked by score, for the shake of simplicity, we will work with nucleotides. On some cases, it can be unfeasible to produce all the possible sequences; the worst scenario is a PWM for a random sequence, where the score of any of the four nucleotides is 0.25, as the number of posible sequences will be 4^n (being n the length of the matrix). For instance a matrix for a nucleotide signal made of 10 nucleotides, n=10, can generate 4^{10} = 1048576 different sequences. As we do not want to fill the disk with too many sequences, we can set a cut-off, say here M=10000 sequences. However, we still must produce only the M sequences that have the highest scores. The simplest way to achieve that is to use a secondary PWM where the positions are reordered by the score of the most frequent nucleotide per position, as well as an array with the original positions, taken as scalar values, recording the order of the new PWM. The best approach can be a recursive function that start generating new sequences iterating over the nucleotides per position sorted from higher frequencies to lower ones. The output records should contain two fields, the generated sequence and the corresponding score, both can be calculated simultaneously. Finally, the script should read a text file containing the PWM in TRANSFAC format, having a section with five fields to store in memory (the ones marked in green, for the position and the relative frequencies for A, C, G and T, in that order). Below you can find two matrices that you have to run separately through your program. Discuss the results obtained from them on the report.

```
AC  M00034
XX
ID  V$P53_01
XX
DE  tumor suppressor p53
XX
BF  T00671; p53; Species: Homo sapiens.
BF  T01806; p53; Species: Mus musculus.
XX
P0      A      C      G      T
01      4      0     13      0      G
02      5      0     12      0      G
03     15      0      2      0      A
04      0     17      0      0      C
05     17      0      0      0      A
06      0      0      0     17      T
07      0      0     17      0      G
08      0     13      0      4      C
09      0     17      0      0      C
10      0     17      0      0      C
11      0      0     17      0      G
12      0      0     17      0      G
13      2      0     15      0      G
14      0     17      0      0      C
15     17      0      0      0      A
16      0      0      0     17      T
17      0      0     17      0      G
18      0      2      0     15      T
19      0     13      0      4      C
20      0      7      2      7      Y
XX
//
```

```
AC  M00097
XX
ID  V$PAX6_01
XX
DE  Pax-6
XX
BF  T00681; Pax-6; Species: Mus musculus.
BF  T01122; Pax-6; Species: Homo sapiens.
XX
P0      A      C      G      T
01     15      7      6     10      N
02     21      9      3     10      N
03     10      9     10     18      N
04      8     14      9     16      N
05      3      2      4     38      T
06      2      0      1     44      T
07      3     29      1     14      C
08     40      5      1      1      A
09      3     39      0      5      C
10      1      0     44      2      G
11      1     36      7      2      C
12     23      2      1     21      W
13      1      4      0     42      T
14      2     13     26      3      G
15     40      1      6      0      A
16     14     11     15      7      N
17      2      4      3     37      T
18      1      0     20     25      K
19     13     17      9      4      N
20     14      8      4      6      N
21      4     12      3      9      N
XX
//
```

GRADES		
	Basic implementation (*working bug-free script*)	**8.0** pt
	Using functions	**+ 1.0** pt
	Report describing implementation and results	**+ 1.0** pt

Last modified: Friday, October 23rd, 2015 @ 10:56:56 am