UNIVERSITY OF TORONTO

FCBII, Assignment 1

Date:	17/05/2025
Course:	MMG1344H
Name:	Maria Eleni Fafouti
Student number:	1010799094

Introduction

For this assignment, we are required to detect motifs across 5 input .txt files, with multiple sequences in each, using 4 models. These models have a similar way of function which is as follows: Each model iteratively searches for a conserved motif of fixed length K (K=6 for this assignment) within each sequence by estimating motif starting positions (offsets) and refining a position frequency matrix (PFM) that represents the motif. In more detail, the PFM represents the probability of encountering each of the 4 RNA nucleotides (A, C, G, U) at a given position in the K-mer motif.

This is done through an Expectation-Maximization-like procedure, where the motif model is updated based on current offsets (M-step), and offsets are re-evaluated based on the motif model (E-step), until convergence or a maximum number of iterations is reached. Each iteration selects one K-mer or a probability distribution of K-mers, depending on the model:

- **HardOOPS** deterministically selects the single highest-scoring K-mer from each sequence based on the current PFM.
- **HardZOOPS** behaves similarly but allows for the possibility that some sequences may not contain a motif at all.
- **GibbsZOOPS** instead samples motif positions probabilistically using weights derived from the log-likelihood under the PFM, allowing for stochastic updates.
- **AnnealZOOPS** follows a similar sampling approach but includes a temperature parameter to gradually reduce randomness and focus on high-probability K-mers over time, similar to simulated annealing.

Method

To write my program, I chose to define a handful of parsimonious functions at the beginning which I will use later in the functions for each of the 4 models. The table below summarizes all the general functions I defined and their purpose:

Function Name	Purpose
read_sequences()	Reads an input file and gathers all valid sequences into a list.
initialize_random_offsets()	Initializes random 1-indexed starting positions for motif extraction in each sequence.
extract_kmer()	Extracts a K-mer from a sequence using a 1-indexed offset (adjusted to 0-indexed slicing).
sample_from_distribution()	Samples an index from a list of probabilities, simulating a discrete random choice according to the provided distribution.
build_pfm()	Constructs a Position Frequency Matrix (PFM) from given motif positions (offsets) across sequences, counting base occurrences with pseudocounts.
score_kmer()	Scores a K-mer by computing the log-likelihood ratio under the PFM versus a uniform background model (log-odds score).
compute_log_e()	Computes the total log-likelihood of selected K-mers under the current PFM, evaluating how well the model fits.
print_pfm()	Nicely prints the PFM as a probability table with each row showing the distribution of a nucleotide across motif positions.
plot_log_e_history()	Plots and saves a line graph showing the progression of log-likelihood values (log E(M, o)) over model iterations.

These functions will enable me to write cleaner code for each model, as well as standardize the output. Finally, since the program will loop over all the input files, it is much more consistent and easier to debug when including functions.

HardOOPS model

I defined a function to run this model which takes as input the sequences we read at the beginning, the value of K (which is defined to be 6 for this assignment) and a value for the maximum iterations which defaults to 100. It works as follows:

First, it initializes the motif offsets randomly in each sequence. It stores the log-likelihood after each iteration in a list to track convergence. For every iteration, it builds the PFM from the current offsets. Then, it loops over each input sequence and tries every possible k-mer position, and computes its score. The k-mer score and its offset are saved and the offsets are updated. Then, it calculates the log E (M,o) of the current motif model and offsets and saves it in the list initialized at the start. After the final iteration, the PFM is rebuilt one last time, using the final offsets.

After running this model, I got the following output for each of the 5 input files:

HardZOOPS model

I defined a function to run this model which takes as input the sequences we read at the beginning, the value of K (which is defined to be 6 for this assignment) and a value for the maximum iterations which defaults to 100. It works as follows:

It runs the HardZOOPS model: each sequence may or may not contain a motif. First, it initializes random offsets, while recording whether a sequence has a motif or not (presence). It also initializes a list to hold the log E values. Then, it runs the E-M loop in a similar way to HardOOPS, but it ensures to update the presence variable. The PFM is built based on the sequences believed to contain a motif. It loops over all possible substrings of length K and finds the best scoring one according to the current PFM. It then records the best k-mer and its offset. Finally, log E is computed using only motif-containing sequences.

GibbsZOOPS model

I defined a function to run this model which takes as input the sequences we read at the beginning, the value of K (which is defined to be 6 for this assignment) and a value for the maximum iterations which defaults to 100. It works as follows:

It runs the GibbsZOOPS model using Gibbs sampling. It initializes the offsets and tracks which sequences have a motif (similarly to the previous ZOOPS model). It also initializes a list to hold the log likelihoods. Then, for each iteration it performs the following: it leaves one sequence out to remove its current motif offset. It then uses the other sequences to build a PFM and then it uses the held-out sequence to score all possible k-mers and then convert those scores into probabilities. Based on those probabilities, it samples a new offset. Before selecting a new start position from the distribution, it adds a flat baseline probability to account for the possibility that a sequence does not have a motif. Now the model can either sample a start position or 'no motif' from the distribution. If the latter happens, the offset is assigned the value 0 and there is no motif. If the sampled index is > 0, then the offsets are updated accordingly. Finally, log E is computed using only motif-containing sequences.

AnnealZOOPS model

I defined a function to run this model which takes as input the sequences we read at the beginning, the value of K (which is defined to be 6 for this assignment) and a value for the maximum iterations which defaults to 100. It works as follows:

It runs the AnnealZOOPS model using simulated annealing. Similar to GibbsZOOPS, this model assumes that each sequence contains at most one motif (ZOOPS) and tracks motif offsets and which sequences contain motifs. It also initializes a log E history list to track progress over iterations.
The algorithm starts with a high temperature and gradually cools it down over iterations. At each step, it proposes small random changes to the motif offsets. This may include modifying the start position of a motif in a sequence, or toggling whether a sequence is considered to contain a motif at all. The new configuration is accepted if it improves the overall log likelihood (log E), or probabilistically based on the current temperature if the log likelihood is worse. This allows exploration of the search space and avoids getting trapped in local optima early on.

As the temperature decreases, the model becomes more selective, eventually settling into a high-scoring motif configuration. The log E score is computed only on motif-containing sequences in each iteration. The final PFM and motif offsets reflect the best configuration encountered during the annealing process.

Running all models on the test files

To automate the analysis, I implemented a loop was implemented to run all four models on each of the five input .txt files. For every input file, the script first reads the RNA sequences, then creates a dedicated output folder for each model. Each model is then applied to the sequences to estimate motif positions (offsets) and construct a position frequency matrix (PFM) through iterative optimization. The output of each model, including runtime details, final log E(M, o) values, estimated offsets, and the resulting PFM, is printed to an output.txt file stored within the corresponding model's folder. Additionally, the log E(M, o) values across iterations are visualized in a line plot and saved as a .png file in the same folder for downstream comparison.