HW1 Stochastic Processes

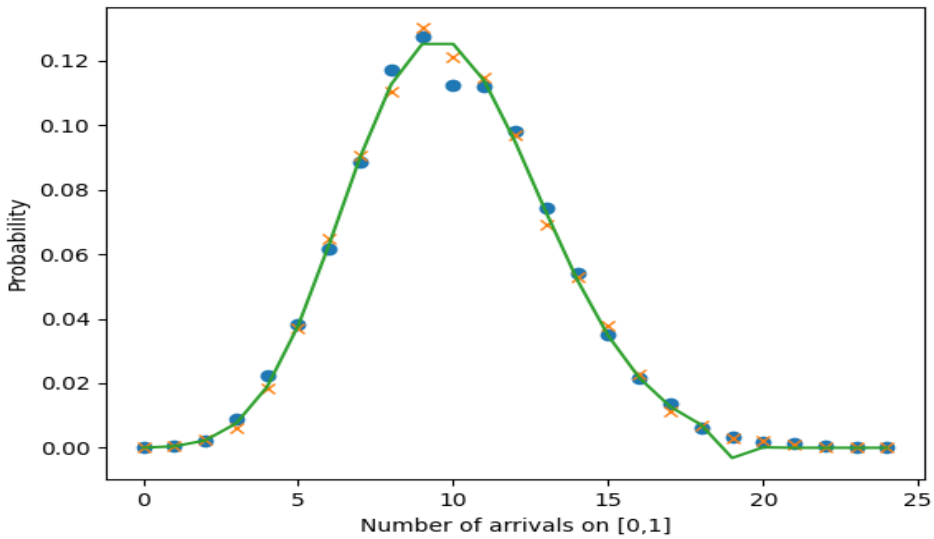
1)

a) In this method, a Poisson process on [0,1] with rate λ is simulated using the defining property that inter-arrival times are i.i.d. exponential random variables with mean 1/λ1. Arrival times are obtained by cumulatively summing these inter-arrival times until the sum exceeds 1. All arrival times less than or equal to 1 are recorded.

b) This method uses the fact that the total number of arrivals in [0,1] follows a Poisson(λ) distribution. Conditional on having N=n arrivals, the arrival times are distributed as the order statistics of n i.i.d. Uniform(0,1) random variables. Therefore, the arrival times are generated by sampling uniform random variables and sorting them.

c) Both Poisson1 and Poisson2 were simulated for 10,000 independent runs with λ=10 and the total number of arrivals in each run was recorded. The empirical distribution of the number of arrivals was plotted and compared with the theoretical Poisson(10) distribution. The close agreement between empirical and theoretical results validates the correctness of both simulation methods.

d) Both methods generate statistically identical Poisson processes and produce the same distribution for the number of arrivals. Poisson1 follows the physical construction of the process via exponential inter-arrival times, while Poisson2 is computationally more efficient by generating the number of arrivals first. Despite different simulation approaches, both methods are theoretically equivalent



2)

a) A Markov Chain assumes that the future depends only on a limited history, not the entire past.

For a character-based k-order Markov model:

$$P(x_{t+1}|x_t, x_{t-1}, \ldots x_{t-k-1})$$

In the character-based Markov chain used for text generation, a state is defined as the last k consecutive characters generated by the model. The choice of the next character depends only on this state, which reflects the Markov property (limited memory).

The state space consists of all distinct substrings of length k that appear in the training text (Hafez poems). Each state has transition probabilities to possible next characters, estimated from their frequencies in the training data.

k = 1: Each state is a single character (first-order Markov chain).

k = 4: Each state is a sequence of 4 characters.

k = 10: Each state is a sequence of 10 characters.

b) completing jupyter notebook

c) Generating text for k = 1,4,10,20 in jupyter notebook.

Analyzing the differences : k = 1: Text is very random; mostly gibberish letters. High creativity but very low fluency. No coherent words or phrases appear.

k = 4: Some local structure appears; short words and partial phrases start to form. Medium fluency and moderate creativity.

k = 10: Text becomes fluent; longer sequences and real words appear. Style of Hafez is recognizable. Creativity decreases as sequences may be repeated from the training text.

k = 20: Almost memorized lines; long sequences closely match the original poems. Very high fluency but very low creativity.

Overall observation: Increasing k improves fluency and grammatical coherence but reduces originality. Small k produces more random, creative text; large k produces almost memorized text with low novelty.

d)

Disadvantages of the character-based Markov chain model: It captures only local dependencies limited to the previous k characters; long-range structure and meaning are lost. For small k, generated text is mostly random and often nonsensical. For large k, the model tends to memorize training text, producing exact sequences and reducing creativity. It cannot understand grammar, semantics, or context; it only learns statistical patterns of characters.

how to improve :
Ways to improve the model: Increase k carefully to balance fluency and creativity, or use variable-length n-grams. Move from character-level to word-level Markov chains to capture more meaningful patterns. Use smoothing techniques to handle unseen k-grams and avoid producing empty or repetitive text. Switch to neural network models (like RNNs, LSTMs, or Transformers) to learn long-range dependencies, grammar, and semantic structure.

3)

a) baum_welch algorithm is implemented to calculate HMM parameters. HMM parameters are saved in a text file.

b) log-probability shows how likely your trained HMM thinks this entire test sequence was generated by the model.

The more negative the value, the less probable the sequence generated by the model.

The less negative log-probability → sequence is more compatible with the model.

The result is saved in a JSON file.

c) The Viterbi algorithm assigns, for each observation in the sequence, the most probable hidden state. The output is a time-aligned sequence showing how observations are explained by hidden states.

The result is saved in a JSON and TEXT format.

d) challenges and their solutions:

1. Numerical underflow
Probabilities shrink exponentially with sequence length.
Solution: scaling in forward–backward and log-space in Viterbi.

2. Forbidden transitions
Some state transitions are impossible by model design.
Solution: introduce a transition mask and enforce it during initialization, Baum–Welch updates, and Viterbi decoding.

3. Baum–Welch instability
Zero probabilities may appear during re-estimation.
Solution: masked normalization and ε-stabilization.

4. Variable-length sequences
Training and test samples have different lengths.
Solution: algorithms implemented without assuming fixed length.

5. State consistency
State indices must match the diagram exactly.
Solution: fixed indexing and explicit transition rules.

It takes too much time to run 10 iterations ;(