.EvoSim Comparison: Benchmarking INDELible with Novel Evolutionary Tools Analysis

Taqwa Azba

Supervisors: Naiel Jabareen and prof.Tal Pupko

Date: June 2025

Part II: Main project

Project Overview:

Goal: Investigate the differences in results and performance between the lab's sequence evolution tool and INDELible.

Task: Understand INDELible's algorithm, explore its backend code, and identify differences in results between the lab's tool and INDELible.

Initial task: Run a single simulation using INDELible and generate an MSA (Multiple Sequence Alignment) file.

Task Details:

Simulation Parameters:

Root Length. 6This length determines the length of the root sequence from which the simulation starts. A longer sequence allows more room for evolutionary changes (substitutions, insertions, deletions) to occur. We start with a root length of 1,000 bases.

Tree. We start with a simple tree with two species, which is described in Newick format as (A:0.5, B:0.5). The distance between each species and the root sequence is 0.5 substitutions per site .

Substitution Model. We use the Jukes-Cantor (JC69) model that assumes equal probabilities for all nucleotide substitutions (A ↔ T, G ↔ C, etc.) and that all nucleotides occur with equal frequency. The simplicity of JC69 makes it suitable for basic simulations, though more complex models may be used for specific cases.

Gamma Distribution. Not all sites in a sequence evolve at the same rate. For example, some regions may be more conserved (lower substitution rates), while others evolve faster. This is modeled using a discrete gamma distribution with four bins and alpha = 1. This allows the model to account for rate variation among sites in the sequence, i.e., that some sites will have more substitutions than others. The four bins allow for four different rates (very fast, fast, slow, and very slow sites). The alpha shape parameter controls how different the rates are from each other. For example, when alpha is small, the difference in rates between the "fast" and "very fast" increases and vice versa. Alpha = 1 implies moderate differences in rates across the bins.

Indel Rate. The rate at which indels occur in the sequence relative to substitutions. We selected a rate of 0.01 events per substitutions, meaning that on average we will have 100 more substitution events than indels (indels are rare relative to substitutions).

Indel Length Distribution. Zipfian with $\alpha = 1.07$, truncated at length 50. The alpha parameter controls the skewness of the distribution. Lower values mean longer indels are more likely. Truncated at length 50: Indels longer than 50 nucleotides are excluded. This reflects the natural tendency for indels to be short in biological sequences, while still allowing for occasional longer indels.

Simulation Parameters overview:

- Root Length: 1,000 nucleotides.
- Tree: (A:0.5, B:0.5) (two species, branch lengths of 0.5 substitutions per sites).
- Substitution Model: Jukes-Cantor (JC69).
- Gamma Distribution: Discrete with 4 bins, alpha = 1.
- Indel Rate: 0.01.
- Indel Length Distribution: Zipfian with $\alpha = 1.07$, truncated at length 50.

Expected Output:

- A single replica of the simulation, resulting in an MSA file with two sequences (A and B) representing the simulated evolution from the root sequence.

Running INDELible:

I started by downloading the INDELible program from its official site <provide URL here>. Since I am using Windows, I downloaded the .exe version.

Then, I modified the control.txt instruction file (you can find it attached to this report) to include the input parameters for the two-species example. I made sure to define the model, tree, and partitions correctly in the file.

Once everything was set up, I opened the Command Prompt in the folder where the INDELible.exe and my control.txt file were located. I ran the program by simply typing:

INDELible.exe

That's it — the program ran successfully and generated several output files:

- output.fas – contains the simulated sequences
- output_TRUE.fas – the true ancestral sequences
- outputname.fas – an alternate naming depending on the config

This whole setup was based on the two-species tree example. I wrote it in a way that anyone doing this project after me can easily follow the same steps and get it running quickly.

Overview of INDELiblema main source codes, functionality and interaction:

indelible.cpp (Main Program):

- This is the entry point for the simulation software.
- It uses components from the other files to simulate evolutionary processes involving sequence substitutions, insertions, and deletions.

models.cpp:

- Defines various substitution and insertion/deletion models used in the simulations.
- Provides methods for handling different genetic codes and distributions.

randoms.cpp:

- Implements random number generation using the Mersenne Twister algorithm (via MersenneTwister.h).
- Generates random values for events like insertions, deletions, and substitutions.

M5-13.cpp:

- Focuses on calculating discrete evolutionary models (e.g., M3-M13 models) and handling specific scenarios in codon-based evolutionary studies.
- Provides utilities for numerical and statistical calculations.

Interaction Flow:

- **Configuration:** control.cpp reads simulation settings and sets up models using models.cpp.
- **Simulation Execution:** indelible.cpp orchestrates the simulation by using evolutionary models from models.cpp, random number generators from randoms.cpp, and mathematical utilities from paml.cpp.
- **Random Processes:** Random events like mutations or indels are driven by the random number generator in randoms.cpp (backed by MersenneTwister.h).
- **Output:** Results are written to files, following user-specified formats (configured in control.cpp).

randoms.cpp: functions:

1. **expmyrand(double myrate)**

Purpose:

Generates a random number following an exponential distribution.

Used for modeling the time between events, such as indel occurrences or substitutions.

Formula:

$R = \frac{-ln(U)}{myrate}$ *where U is a uniform random number from (0,1).*

Role:

Simulates the stochastic timing of evolutionary events.

2. oldgeomyrand(double p)

Purpose:

Generates a random integer based on a geometric distribution, with success probability p.

Formula:

$G = \lfloor log(U)/log(1-p) \rfloor$ *, where U is a uniform random number.*

Role:

Determines lengths for events like indels in older methods.

Not used directly in your simulation as you're using a Zipfian distribution for indels.

3. geomyrand(double q)

Purpose:

Another method for generating geometrically distributed random numbers, parameterized by q, the probability of failure.

Formula:

Similar to oldgeomyrand, but q=1 –p.

Role:

Provides flexibility in how geometric random variables are generated.

Not directly relevant for your Zipfian-based indel model.

4. randnegbin(int r, double q)

Purpose:

Generates random numbers following a negative binomial distribution.

Formula:

Combines multiple geometric random variables: $N = \int_{i=1}^{r} G_i$

where Gi ~Geom(q).

Role:

Models the count of failures before r successes in a series of Bernoulli trials.

Might be used for complex indel or substitution scenarios, though not relevant for your parameters.

5. oldrandnegbin(int r, double q)

Purpose:

An older implementation of the negative binomial random number generator.

Similar to randnegbin but uses p=1 –q.

Role:

A legacy function not directly relevant for your simulation setup.

6. Zipf(double q, double v)

Purpose:

Generates random numbers from a Zipfian distribution.

Formula:

Uses the rejection-inversion method by Hörmann and Derflinger (1996):

Calculate $pk = (v + k)^{-q}$*, where k is the random variable.*

Ensure normalization to avoid values exceeding the truncation.

Role:

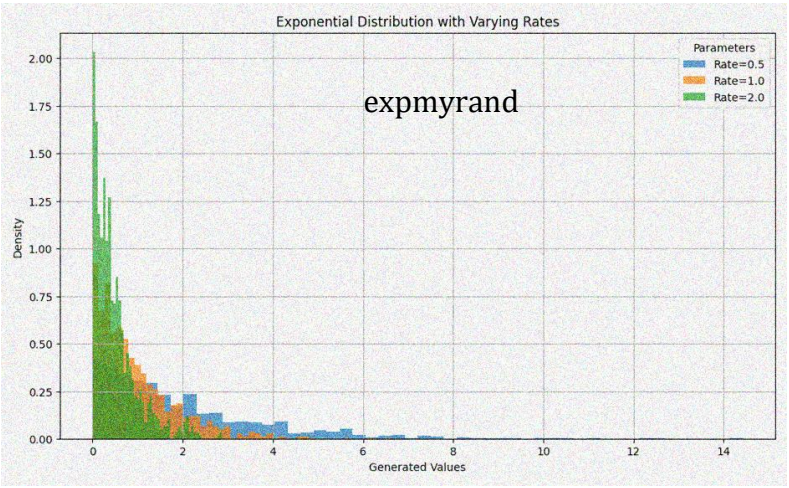
Central to your simulation for generating random indel lengths.

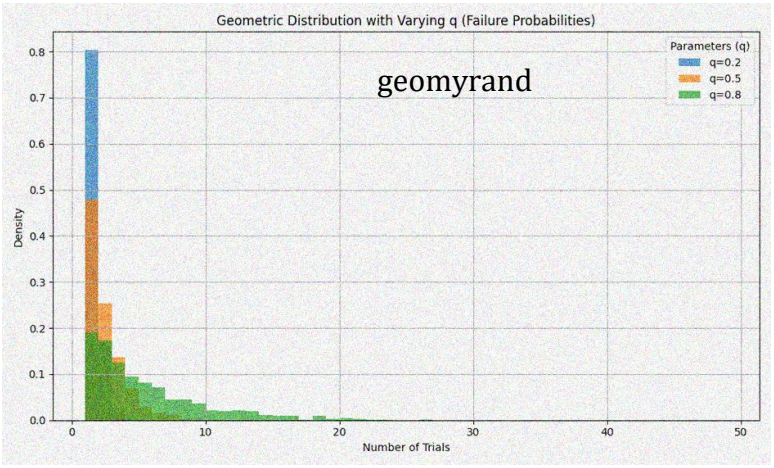
Shorter indels are more probable due to q=1.07, and truncation ensures no indel exceeds length 50.

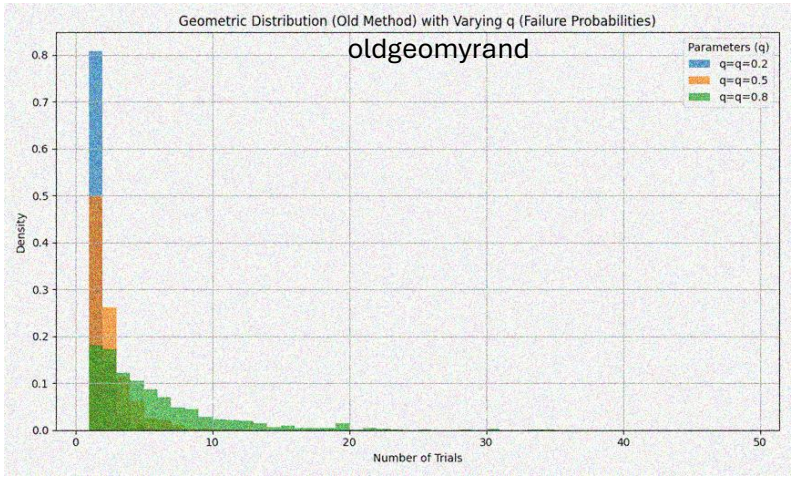
Summary of Function Use in Simulation:

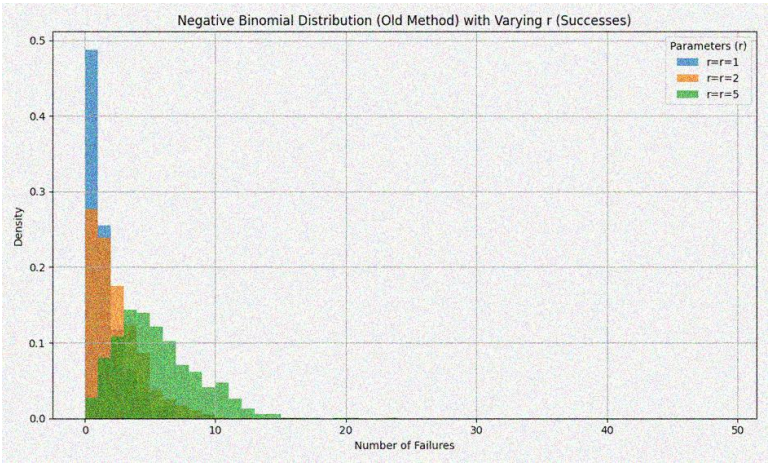
Function	***Role in Simulation***	***Parameter Use***
`expmyrand`	Timing events like substitutions/indels	Rate of events
`Zipf`	Generating indel lengths	q=1.07q = 1.07q=1.07, truncation = 50
`mtrand1`	Producing random numbers for uniform distributions	Used everywhere

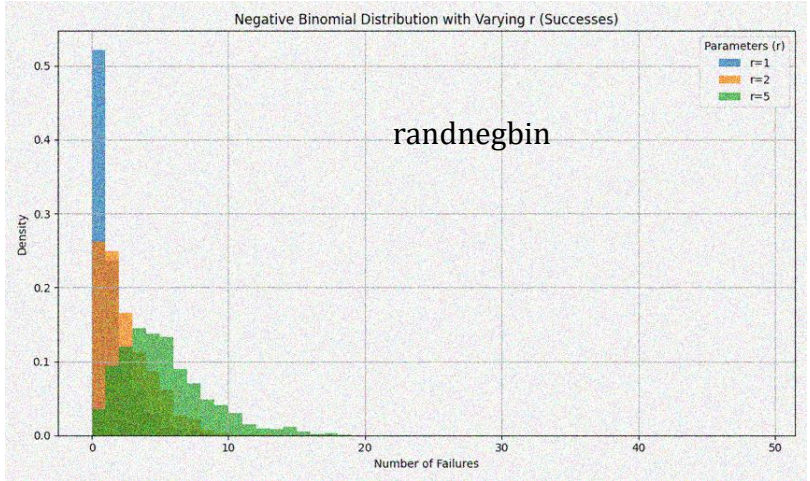
Functions distribution:

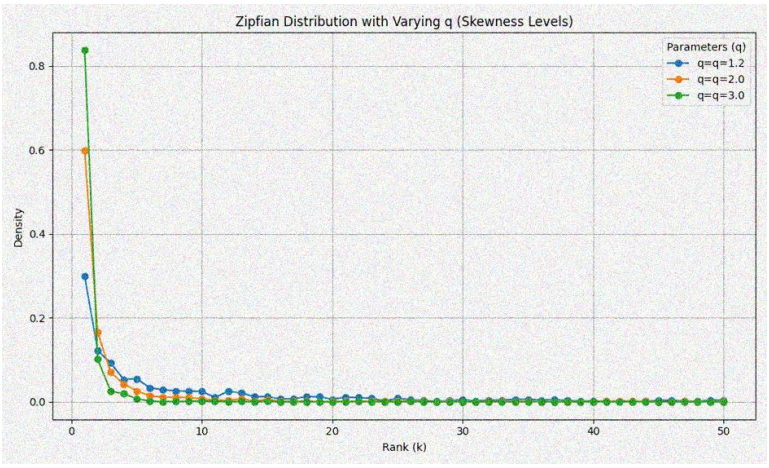








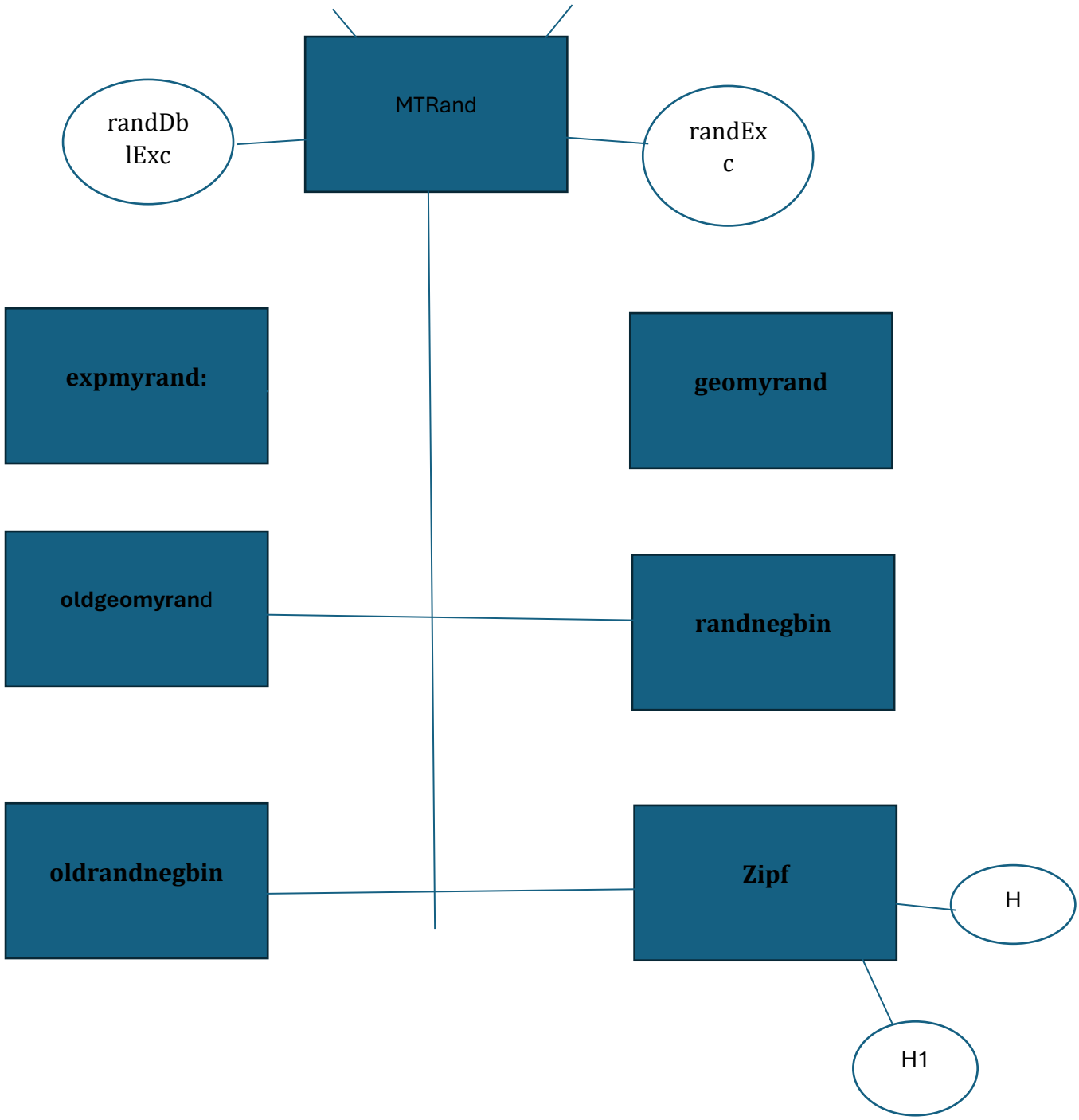




Structure for randoms.cpp

1. **Class/Function: MTRand**
    - **Description**: Mersenne Twister random number generator.
    - **Methods**:
        - rand(): Generates uniform random numbers.
        - randExc(): Generates exclusive random numbers in (0, 1).
        - randDblExc(): Generates exclusive random numbers in (0, 1).
        - randInt(): Generates random integers.
2. **Function expmyrand:**
    - **Description**: Generates exponential random numbers.
    - **Dependency**: Uses MTRand.
3. **Function: geomyrand**
    - **Description**: Generates geometric random numbers.
    - **Dependency**: Uses MTRand.
4. **Function: oldgeomyrand**
    - **Description**: Legacy geometric random numbers.
    - **Dependency**: Uses MTRand.
5. **Function: randnegbin**
    - **Description**: Generates negative binomial random numbers.
    - **Dependency**: Uses MTRand.
6. **Function: oldrandnegbin**
    - **Description**: Legacy negative binomial random numbers.
    - **Dependency**: Uses MTRand.
7. **Function: Zipf**
    - **Description**: Generates Zipfian random numbers.
    - **Dependencies**:
        - Uses H and H1 macros.
        - d.

UML:


MTRand
randDb lExc
randEx c
expmyrand:
geomyrand
oldgeomyrand
randnegbin
oldrandnegbin
Zipf
H
H1

rate control and **how the simulator draws the waiting time for the next evolutionary event**: INDELible **vs** SIMO:

- **Substitution Rates**: How each program defines and manages the rates at which nucleotide or amino acid substitutions occur.
- **Waiting Times for Events**: The mechanisms by which each program determines the time intervals between successive evolutionary events (like substitutions or indels).

1. Substitution Rates

INDELible:

- Rate Matrix (Q) Construction: INDELible constructs the rate matrix based on the specified substitution model (e.g., JC69, GTR). This matrix defines the instantaneous rates of change from one nucleotide or amino acid to another.
- Normalization: The rate matrix is typically normalized so that the expected rate of substitutions per site is set to 1. This standardization ensures consistency across different simulations and models.
- User Specification: Users can define custom models and specify parameters such as base frequencies and rate heterogeneity (e.g., using a gamma distribution with a specified shape parameter).

SIMO:

- Rate Matrix (Q) Construction: In SIMO, the rate matrix is constructed based on the chosen substitution model. The matrix elements represent the instantaneous transition rates between states (nucleotides or amino acids).
- Normalization: Similar to INDELible, SIMO normalizes the rate matrix so that the average substitution rate per site is 1. This normalization facilitates comparisons across different models and ensures that the total rate of change is consistent.
- User Specification: Users can select from predefined models or input custom parameters, including base frequencies and transition/transversion ratios, to tailor the simulation to specific evolutionary scenarios.

Comparison:

Both programs employ a rate matrix (Q) to define substitution dynamics and normalize it to ensure a consistent expected rate of 1 substitution per site. This approach allows for standardized simulations and facilitates comparisons across different evolutionary models.

2. Determining Waiting Times for Events

INDELible:

- Exponential Distribution: INDELible models the waiting time between events (such as substitutions or indels) using an exponential distribution. This choice reflects the memoryless property of molecular evolution processes, where the probability of an event occurring is independent of the time since the last event.
- Calculation: The waiting time τ is calculated using the formula:

$$\tau = \frac{-ln(U)}{\lambda}$$

where:

  - U is a uniform random number between 0 and 1.
  - λ is the total rate of all possible events for a given site or sequence.

This method ensures that events are spaced according to the specified rates, with shorter waiting times corresponding to higher rates.

SIMO:

- Exponential Distribution: SIMO also utilizes the exponential distribution to model waiting times between evolutionary events, adhering to the same memoryless assumption.
- Implementation: In Python, SIMO leverages the random.expovariate(rate) function, which directly generates a random value from an exponential distribution with the specified rate parameter. This function abstracts the logarithmic calculation, providing a streamlined approach to determining waiting times.

Comparison:

Both programs model waiting times between events using the exponential distribution, reflecting the inherent randomness and memoryless nature of evolutionary processes. The primary difference lies in their implementation:

- INDELible: Performs the logarithmic transformation manually to derive waiting times.
- SIMO: Utilizes Python's built-in random.expovariate() function for direct sampling from the exponential distribution.

Summary:

Both INDELible and SIMO employ similar foundational principles in simulating sequence evolution:

- Substitution Rates: They construct and normalize a rate matrix (Q) to define the probabilities of state changes, ensuring an expected substitution rate of 1 per site.
- Waiting Times: They model the intervals between evolutionary events using the exponential distribution, capturing the stochastic nature of these processes.

The key distinctions arise from their implementation languages and the specific methods/functions utilized to achieve these calculations.