A Topic Model-Based Approach to Inferring Episodic Directional Selection in Protein-Coding Sequences

Hassan Sadiq
SDQHAS001

A minor dissertation submitted to the
Department of Statistical Sciences
University of Cape Town, South Africa

In partial fulfillment of the requirements of the degree of
Master of Science

Under the supervision of
Doctor Miguel Lacerda

November 18, 2015.

Abstract

Pathogens, such as HIV and influenza, evolve in response to the selective pressures of their host environments accumulating changes in their genomes that offer fitness benefits. This selective pressure is characterised by three properties: (1.) it is episodic, tracking changes in the adaptive immune response and drug therapy, (2.) it is directional in that only particular amino acid substitutions are favoured and (3.) it varies between genomic loci. Most previous models have ignored or inadequately addressed some of these phenomena. This work extends recent approaches to modelling episodic directional selection acting on protein-coding sequences. We use inference techniques within the topic model framework to identify loci evolving under natural selection. A notable example of such techniques are the variational Bayesian methods. We show that our approach performs well in terms of specificity and power, and demonstrate its utility by applying it to some real datasets of HIV sequences.

Acknowledgements

First, I will like to thank my supervisor, Dr. Miguel Lacerda, for his impeccable guidance throughout this research. I am especially grateful for his time and efforts in providing me with punctual and insightful comments. The quality research etiquettes I learnt from him will continue to guide me in my future academic commitments. He has been an exceptional role model whose keen interest beyond my academic well-being helped me foster an improved sense of self-worth. I am very fortunate to have gained from his expertise and I look forward to working with him in the nearest future.

This dissertation would have been more difficult without the contributions of a few people. I appreciate the answers provided to all of my questions on the `veg/hyphy` user platform by Steven Weaver and A/Prof Sergei L Kosakovsky Pond of the School of Medicine at the University of California San Diego. Andrew Lewis of the ICTS at the University of Cape Town (UCT) selflessly helped with the intense computational requirements of this work. Some technical comments passed by my anonymous examiners contributed immensely to the quality of this final draft of the dissertation and I am sincerely grateful.

I am highly indebted to my parents and all my family members for supporting me through my studies. I will like to recognise the extraordinary contributions of my sister, Ms. Rissicatou. Her dedication to my success are beyond words. I will never forget how she ensured I was well taken care of despite the 2011 turmoil in her resident country, Cote d'Ivoire. The contributions of my brother, Abdel-Liazize, are also worthy of special mention. I acknowledge the undying support and the words of encouragement from my twin brother and confidant, Husseyn Sadiq. The faith they all had in me helped me get through this research.

I thank all the staff of the Department of Statistical Sciences at UCT for their various contributions towards ensuring that the Department remains an ideal research environment. I will like to specially thank Professor Linda Haines for guiding me through my first research work and her continuous support since then. A/Prof Sugnet Lubbe has always been my pillar of support since my early days at the university. I thank her because the knowledge that she is always available to listen and help has always granted me some special feeling of safety and allowed me to freely take on various academic commitments that helped in completing this dissertation. I deeply appreciate the special interest shown by A/Prof Francesca Little in ensuring the successful completion of this work and her tolerance of my extortion of her kind office.

I am grateful to all my class of 2014 Statistics Masters colleagues, most importantly Ms. Raeesa, Nashruddin and Tosin, for being part of making my journey filled with many pleasant and unforgettable moments. I will like to thank all my friends that made South Africa a home for me, notably Réne Glass, Mpumelelo Kondlo, Phaswana Awelani, Abigail Osiki, Riaz Patel, Sa'addiya Mohammed and the Ganeys. Finally, I thank God for His mercies.

Hassan Sadiq
November 7, 2015.

Contents

Chapter 1

Introduction

The development of an effective vaccine against a pathogen requires a sound understanding of how the pathogen evolves to avoid the immune response and drug treatment. This is especially true for rapidly evolving viruses such as the type 1 human immunodeficiency virus (HIV-1). Antiretroviral therapies against HIV-1 accelerate the rate at which mutations that impart drug resistance become established in the intra-host viral population (Novak et al., 2005; Clavel and Hance, 2004). Identifying the genomic positions that evolve under this strong selective pressure is important for vaccine development.

Many evolutionary models have been developed to identify amino acid residues that evolve under positive selection (Quang et al., 2008; Massingham and Goldman, 2005; Kosakovsky Pond and Frost, 2005b; McDonald and Kreitman, 1991). However, most of these do not permit the selective pressure to vary in strength over time. In HIV transmission chains, for example, drug resistance-associated mutations only offer a selective advantage when the virus is exposed to the drug. Recently, Murrell et al. (2012a) developed a model to detect sites in protein-coding sequences that evolve under such episodic positive selection.

This minor dissertation extends the work of Murrell et al. (2012a). It developed a computationally efficient statistical framework for inferring mutations that act on amino acid sequences, vary in strength over time and are accelerated towards a small set of amino acids conferring drug resistance. Ideas from the field of information retrieval were employed for this purpose, in particular the latent Dirichlet allocation model of Blei et al. (2003).

The literature on the models of evolution is reviewed in the next chapter. The methods and inference procedures are presented in Chapter 3. Chapter 4 is dedicated to presenting findings from the simulation studies that were carried out to investigate the performance of the models on data generated under

various conditions. The most effective inference routine was then applied to three empirical HIV-1 datasets and the results are discussed in Chapter 5. The conclusions of this minor dissertation and suggestions for future research are presented in Chapter 6.

Chapter 2

Literature review

The differences and similarities between protein sequences provide information about the forces of evolution that have acted on the sequences since they diverged from their most recent common ancestor (MRCA) (Cavalli-Sforza and Edwards, 1967). Modelling these changes over time is an important goal in evolutionary biology. Evolutionary relationships are often represented with bifurcating trees referred to as phylogenies. This chapter explains the techniques used to model amino acid changes in protein coding sequences along a phylogeny.

2.1 Models of amino acid substitution

Evolutionary models are convenient simplifications of a complex biological reality that are used to draw inferences about the substitution events along an evolutionary tree (Liò and Goldman, 1998). There are different kinds of evolutionary models that differ in terms of the assumptions they impose. Models of amino acid evolution are discussed in this section. The fundamentals of amino acid evolutionary models in the context of two protein coding sequences, are first presented. Then, the concepts are extended to multiple sequences. In closing, the different types of amino acid models are explained.

2.1.1 Modelling the evolution of a pair of protein-coding sequences

In this subsection, the principal focus is on the analysis of an alignment of two protein coding sequences. Consider the aligned protein coding sequences

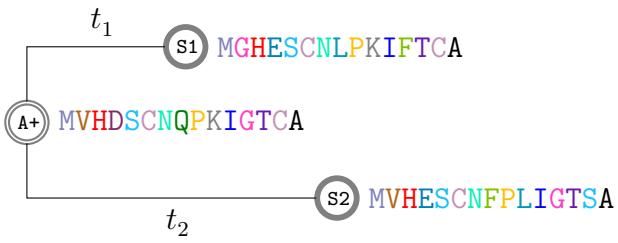
of two taxa `S1` and `S2` that share a MRCA presented in Figure 2.1(a). Recency in this context implies that the sequences still possess enough evolutionary information so that they can be aligned. Figure 2.1(b) presents the sequences on a phylogeny, including the unobserved ancestor sequence (`A+`). The differences between these sequences represent signatures of events that have altered the residues since they diverged (Stark et al., 2007).

A conventional measure of the degree of divergence between homologous protein coding sequences is the expected number of amino acid substitutions per column (i.e. site) of the sequence alignment. This measure is referred to as the genetic distance, d. A simple estimate of d may be computed as the proportion of non-matching (i.e. polymorphic) sites. In Figure 2.1, there are $m = 5$ polymorphic sites among the $N = 15$ total sites. Therefore, the p-distance, $\hat{p} = 5/15$ is an estimate of d. The p-distance gives a reasonable estimate of d only when the time of divergence between sequences is small enough for it to be highly unlikely that any site may have experienced multiple substitutions. As exemplified in Table 2.1, although sites 2 and 18 of the alignment only experienced a single substitution event, other sites had multiple substitutions that resulted in an underestimation of d by the p-distance.

The effect of multiple substitutions on the estimation of genetic distance is usually accounted for with a continuous time Markov chain (CTMC) probabilistic generative model. In the case of amino acid models, the state space, $\mathcal{S}$, is the 20 amino acid residues. The suitability of CTMCs for this distance adjustment task is established by the Chapman-Kolmogorov equations. Ross (2010) presents thorough explanations and proofs of the equations that will be encountered below. Yang (2006) presents a similar discussion in a phylogenetic context.

```
S1   MGHESCNLPKIFTCA
S2   MVHESCNFPLIGTSA
```

(a) Sequence alignment.



(b) Phylogeny.

Figure 2.1: Alignment and phylogeny of two observed homologous protein coding sequences collected at a particular time (and the unobserved ancestor sequence, `A+`). The branch lengths, t_1 and t_2, represent the expected number of substitutions since the associated species diverged from `A+`.

Table 2.1: Illustration of how genetic distance, d, may be underestimated by p-distance, $\hat{p}$, in the presence of multiple substitutions. `S1` and `S2` are the observed sequences and `A+` is their most recent common ancestor.

(a.) p-distance

S1	S2	Polymorphic site
M	M	
G	V	1
H	H	
E	E	
S	S	
C	C	
N	N	
L	F	1
P	P	
K	L	1
I	I	
F	G	1
T	T	
C	S	1
A	A	
m:		5
$\hat{p} = 5/15$		

(b.) **Observed genetic distance**

S1	Hidden substitutions (from A+)	S2	Substitution count
M	M	M	
G	G ← V	V	1
H	H	H	
E	E ← D → E	E	2
S	S ← Q ← S	S	2
C	C	C	
N	N	N	
L	L ← Q → F	F	2
P	P	P	
K	K → H → L	L	2
I	I	I	
F	F ← E ← G	G	2
T	T	T	
C	C → S	S	1
A	A	A	
Total:			12
$d = 12/15$			

Let $p_{ij}(t)$ be the transition probability from state $i \in \mathcal{S}$ to state $j \in \mathcal{S}$ after divergence time $t = t_1 + t_2$. Here, t_1 and t_2 represent the time that taxa `S1` and `S2` evolved independently since they diverged from their common ancestor. According to the representation used in Figure 2.1(b), branch lengths are read as horizontal extensions. The vertical space between the branches is solely illustrative and has no useful interpretation. The Chapman-Kolmogorov equations state that

$$p_{ij}(t) = \sum_{k \in \mathcal{S}} p_{ik}(t_1) p_{kj}(t_2). \tag{2.1}$$

This implies that $p_{ij}(t)$ accommodates all the possible routes that may have been explored in the transition from i to j in time t. For example, assume time reversibility holds for the illustration in Figure 2.1. At site two in the figure i represents glycine (`G`), j is valine (`V`) and k denotes all the unobserved substitutions that may have occurred between these residues in time $t_1 + t_2$. Henceforth, the matrix of transition probabilities at time t will be represented

as $\mathbb{P}_t = \{p_{ij}(t)\}$ with a boundary condition that states that $\mathbb{P}_0 = \mathbf{I}$, where $\mathbf{I}$ is a 20×20 identity matrix.

Let $\mathbb{Q} = \{q_{ij}\}$ represent the instantaneous substitution rate matrix such that q_{ij} is the rate of change in the probability that amino acid $i \in \mathcal{S}$ is substituted with amino acid $j \neq i \in \mathcal{S}$ in an infinitesimally small amount of time, Δt. Mathematically,

$$q_{ij} = \lim_{\Delta t \to 0} \frac{\Pr\big[x(t + \Delta t) = j | x(t) = i\big]}{\Delta t},$$

where $x(t)$ is the state of the CTMC at time t.

The CTMCs that are commonly used to model amino acid evolution typically have non-zero substitution rates i.e. $q_{ij} > 0$ for all $i \neq j$. Their rows are constrained to sum to zero. Since the off-diagonals are positive, it follows that the diagonals are given as

$$q_{ii} = -\sum_{i \neq j} q_{ij},$$

for $i, j \in \mathcal{S}$. Intuitively, $-q_{ii}$ represents the rate of substitution away from residue i. Observe the difference between q_{ij} and $p_{ij}(t)$. Transition probability, $p_{ij}(t)$, gives the probability of amino acid $i \in \mathcal{S}$ being substituted with amino acid $j \in \mathcal{S}$ after time $t > 0$. That is

$$p_{ij}(t) = \Pr[x(t) = j | x(0) = i].$$

By solving the Kolmogorov equations, it can be shown that $\mathbb{P}$ and $\mathbb{Q}$ are related by the following differential equation

$$\frac{\mathrm{d}\mathbb{P}_t}{\mathrm{d}t} = \mathbb{P}_t \mathbb{Q}.$$

The solution to the differential equation is given by

$$\mathbb{P}_t = \exp\{\mathbb{Q}t\} = \sum_{a=0}^{\infty} \frac{(\mathbb{Q}t)^a}{a!}. \tag{2.2}$$

Let $\boldsymbol{\pi}_t = (\pi_t^{(\mathtt{A})}, \pi_t^{(\mathtt{C})}, \ldots, \pi_t^{(\mathtt{Y})})$ represent the probability distribution over the state space $\mathcal{S}$ at time t, where $\pi_t^{(\star)}$ represents the frequency of amino acid residue $\star$ at time t and $\sum_{i \in \mathcal{S}} \pi_t^{(i)} = 1$. The vector of equilibrium frequencies, $\boldsymbol{\pi}_t$, is a function of $\mathbb{P}_t$ and can be expressed as follows

$$\boldsymbol{\pi}_t = \boldsymbol{\pi}_0 \mathbb{P}_t.$$

It is noteworthy that the models that are the subject of this dissertation are applicable when the Markov process is initially not at equilibrium. For the purpose of the following explanations, an equilibrium state was assumed. At equilibrium, $\boldsymbol{\pi}_t$ is independent of the starting point such that

$$\boldsymbol{\pi} = \boldsymbol{\pi}\mathbb{P}_t.$$

At equilibrium, the expected number of substitutions per site (i.e. genetic distance, d) after divergence time, t, is

$$d = t\sum_{i\in\mathcal{S}}\sum_{j\neq i}\pi^{(i)}q_{ij} = -t\sum_{i\in\mathcal{S}}\pi^{(i)}q_{ii}. \tag{2.3}$$

It is customary to scale the instantaneous substitution rate matrix, $\mathbb{Q}$, such that $-\sum_i \pi^{(i)}q_{ii} = 1$. This ensures that time is measured in terms of genetic distance, since Equation (2.3) will then simplify to $d = t$.

The instantaneous substitution matrix may be parameterised in different ways. The simplest of these is a model where all substitution rates are assumed to be equal, commonly referred to as the equiprobable (EQU) model (Wilbur, 1985). The most parameter-rich substitution model is one that allows a different substitution rate parameter for each amino acid pair, referred to as the unrestricted (UNREST) model (Yang, 1994) with generator matrix,

$$\mathbb{Q} = \begin{array}{c} \\ \mathtt{A} \\ \mathtt{C} \\ \mathtt{D} \\ \vdots \\ i \\ \vdots \\ \mathtt{Y} \end{array}\!\!\begin{array}{c} \begin{array}{cccccccccccc} \mathtt{A} & \mathtt{C} & \mathtt{D} & \mathtt{E} & \mathtt{F} & \mathtt{G} & \mathtt{H} & \mathtt{I} & \cdots & j & \cdots & \mathtt{Y} \end{array} \\ \begin{pmatrix} q_{\mathtt{AA}} & q_{\mathtt{AC}} & q_{\mathtt{AD}} & q_{\mathtt{AE}} & q_{\mathtt{AF}} & q_{\mathtt{AG}} & q_{\mathtt{AH}} & q_{\mathtt{AI}} & \cdots & q_{\mathtt{A}j} & \cdots & q_{\mathtt{AY}} \\ q_{\mathtt{CA}} & q_{\mathtt{CC}} & q_{\mathtt{CD}} & q_{\mathtt{CE}} & q_{\mathtt{CF}} & q_{\mathtt{CG}} & q_{\mathtt{CH}} & q_{\mathtt{CI}} & \cdots & q_{\mathtt{C}j} & \cdots & q_{\mathtt{CY}} \\ q_{\mathtt{DA}} & q_{\mathtt{DC}} & q_{\mathtt{DD}} & q_{\mathtt{DE}} & q_{\mathtt{DF}} & q_{\mathtt{DG}} & q_{\mathtt{DH}} & q_{\mathtt{DI}} & \cdots & q_{\mathtt{D}j} & \cdots & q_{\mathtt{DY}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ q_{i\mathtt{A}} & q_{i\mathtt{C}} & q_{i\mathtt{D}} & q_{i\mathtt{E}} & q_{i\mathtt{F}} & q_{i\mathtt{G}} & q_{i\mathtt{H}} & q_{i\mathtt{I}} & \cdots & q_{ij} & \cdots & q_{i\mathtt{Y}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ q_{\mathtt{YA}} & q_{\mathtt{YC}} & q_{\mathtt{YD}} & q_{\mathtt{YE}} & q_{\mathtt{YF}} & q_{\mathtt{YG}} & q_{\mathtt{YH}} & q_{\mathtt{YI}} & \cdots & q_{\mathtt{Y}j} & \cdots & q_{\mathtt{YY}} \end{pmatrix} \end{array}.$$

For computational convenience, amino acid substitution models are usually assumed to be reversible. Mathematically, this has the following implications. For any amino acid residues i and j and a given divergence time t,

$$\begin{aligned} \pi^{(i)}p_{ij}(t) &= \pi^{(j)}p_{ji}(t) \\ \mathbb{Q} &= \mathbb{S}\Pi, \end{aligned} \tag{2.4}$$

where $\mathbb{S}$ is a symmetrical matrix of amino acid exchangeabilities (Whelan and Goldman, 2001) and Π is a diagonal matrix of equilibrium frequencies $\left(\pi^{(\mathtt{A})}, \pi^{(\mathtt{C})}, \ldots, \pi^{(\mathtt{Y})}\right)$.

Probabilistic modelling of an aligned pair of amino acid sequences begins with a hypothesis on a model of amino acid substitution $\mathbb{Q}$. Then the set of (unknown) model parameters, $\boldsymbol{\theta} = \{\mathbb{S}, \Pi, \tau, \mathbf{t}\}$, can be estimated. Here, τ represents the structure of the phylogeny (i.e. topology) and $\mathbf{t}$ is the set of divergence times. Given that there is only one possible topology for a pair of sequences (see Figure 2.1), inclusion of τ as part of the model parameters is unnecessary. The introduction of tree topology at this stage is in anticipation of the more general case of multiple sequences that is presented in the next subsection.

Consider Figure 2.1. Let x_n represent the amino acid residues observed at the nth site such that $x_n^{(r)}(t_r)$ is the amino acid residue observed in sequence $r \in \{1, 2\}$ at that site after divergence from `A+` in time t_r, where $n \in \{1, 2, \ldots, N\}$. The probability of observing x_n will be given by

$$\begin{aligned} f(x_n; \boldsymbol{\theta}) &= \Pr\left[x_n^{(1)}(t_1) = i,\, x_n^{(2)}(t_2) = j | \boldsymbol{\theta}\right] \\ &= \sum_{k \in \mathcal{S}} \pi^{(k)} p_{ki}(t_1) p_{kj}(t_2). \end{aligned} \tag{2.5}$$

The sum over all possible amino acids k in the sample space $\mathcal{S}$ is used to account for the unknown ancestor residue y_n. With the assumption of reversibility and by the Chapman-Kolmogorov equation, Equation (2.5) simplifies to

$$\begin{aligned} f(x_n; \boldsymbol{\theta}) &= \sum_{k \in \mathcal{S}} \pi^{(i)} p_{ik}(t_1) p_{kj}(t_2) \\ &= \sum_{k \in \mathcal{S}} \pi^{(i)} p_{ij}(t), \end{aligned}$$

where $t = t_1 + t_2$. This implies that by assuming reversibility, the placement of the root is irrelevant because the likelihood value would be the same irrespective of where the root is placed.

Given the above expression of the probability of observing the residues at a particular site, it is straightforward to obtain the likelihood of the sequence alignment. If it is assumed that the sites in the aligned pair of sequences X evolve independently, the likelihood of the sequences may be calculated as

$$\mathcal{L}(\boldsymbol{\theta} | X) = \prod_{n=1}^{N} f(x_n; \boldsymbol{\theta}). \tag{2.6}$$

Consider the following example of likelihood calculation with respect to the EQU model, where

$$\mathbb{S} = \begin{array}{c} \\ \mathtt{A} \\ \mathtt{C} \\ \vdots \\ i \\ \vdots \\ \mathtt{Y} \end{array} \begin{array}{c} \begin{array}{ccccccc} \mathtt{A} & \mathtt{C} & \mathtt{D} & \mathtt{E} & \dots & j & \dots & \mathtt{Y} \end{array} \\ \begin{pmatrix} -19\varphi & \varphi & \varphi & \varphi & \dots & \varphi & \dots & \varphi \\ \varphi & -19\varphi & \varphi & \varphi & \dots & \varphi & \dots & \varphi \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \varphi & \varphi & \varphi & \varphi & \dots & \varphi & \dots & \varphi \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \varphi & \varphi & \varphi & \varphi & \dots & \varphi & \dots & -19\varphi \end{pmatrix} \end{array},$$

$$\boldsymbol{\pi} = \begin{pmatrix} \frac{1}{20}, & \frac{1}{20}, & \frac{1}{20}, & \frac{1}{20}, \dots, & \frac{1}{20} \end{pmatrix}.$$

The number of substitutions over a fixed divergence time t will be a random variable distributed as Poisson$(19\varphi t)$. From Equation (2.2), the transition probability matrix, $\mathbb{P}_t$, will have elements

$$p_{ij} = \begin{cases} \frac{1}{20} + \frac{19}{20}\exp(-20\varphi t) & \text{for } i = j \\ \\ \frac{1}{20} - \frac{1}{20}\exp(-20\varphi t) & \text{for } i \neq j. \end{cases}$$

Consequently

$$\mathcal{L}(\boldsymbol{\theta}|X) = \left(\frac{1}{20}\left[\frac{1}{20} + \frac{19}{20}\exp(-20\varphi t)\right]\right)^{N-m}\left(\frac{1}{20}\left[\frac{1}{20} - \frac{1}{20}\exp(-20\varphi t)\right]\right)^{m},$$

where m is the number of polymorphic sites among the N sites in the alignment. Observe from the likelihood equation above that φ and t occur together. This implies that it is impossible to estimate these two parameters separately without external information. Thus, the maximum likelihood estimate (MLE) for $\vartheta = \varphi t$ may be computed as

$$\hat{\vartheta} = -\frac{1}{20}\ln\left(1 - \frac{20}{19}\frac{m}{N}\right) = -\frac{1}{20}\ln\left(1 - \frac{20}{19}\hat{p}\right),$$

where $\hat{p}$ is the unadjusted distance estimated from the sequence alignment. The MLE of the genetic distance d may then be calculated by the invariance property of MLEs as

$$\hat{d} = \mathrm{E}[\,\widehat{\mathrm{Poisson}(19\vartheta)}\,] = 19\hat{\vartheta} = -\frac{19}{20}\ln\left(1 - \frac{20}{19}\hat{p}\right).$$

There is, however, more to the analysis of sequences than simply estimating the genetic distance. Phylogenetic analysts are interested in understanding the forces that drive evolution. This might be achieved through a suitable parameterisation of the substitution rate matrix. Such parameterisations will be the focus in subsequent sections. Meanwhile, how the above modelling techniques extend to the analysis of more than two sequences and the types of amino acid models available in the literature are discussed in the next two subsections.

2.1.2 Multiple sequence analysis on a fixed topology

Empirical phylogenetic inferences are commonly drawn from analyses of multiple (more than two) sequences because it is difficult to make reliable inferences otherwise. The derivation of the likelihood function for multiple sequences is a direct extension of the derivation of Equation (2.6). Consider modelling the alignment of four protein sequences in Figure 2.2(a) that are related by the phylogeny in Figure 2.2(b). Figure 2.2(c) is Figure 2.2(b) zoomed in on site n.

On the phylogenies in Figure 2.2, t_r represent the length of branch r. Relative to the two-sequence analyses presented in section 2.1.1, the topology is an important pre-requisite for modelling the sequence alignment and it is typically inferred from the sequence. Felsenstein (2004) provides a treatise on phylogeny reconstruction.

Assume the ancestral sequences $\{\texttt{A0}, \texttt{A1}, \texttt{A2}\}$ are given and that the substitution process governing the evolution of the sequence alignment is in equilibrium. Let τ represent the topology, $\mathbf{t} = \{t_1, t_2, \ldots, t_6\}$ be the vector of branch lengths and define $\boldsymbol{\theta} = \{\mathbb{S}, \Pi, \tau, \mathbf{t}\}$. The probability of observing the amino acids at site n, $x_n = \{\texttt{N},\texttt{P},\texttt{N},\texttt{M},\texttt{N},\texttt{M},\texttt{N}\}$, in Figure 2.2 can then be expressed as,

$$f(x_n;\boldsymbol{\theta}) \;=\; \pi^{(\texttt{N})} p_{\texttt{NP}}(t_1) p_{\texttt{NN}}(t_2) p_{\texttt{PM}}(t_3) p_{\texttt{PN}}(t_4) p_{\texttt{NM}}(t_5) p_{\texttt{NN}}(t_6).$$

In reality, the ancestral sequences are rarely known. That is, $x_n = \{\texttt{M},\texttt{N},\texttt{M},\texttt{N}\}$. This research is designed to cater for such cases. Like Equation (2.6), to account for the uncertainty surrounding the unknown ancestral sequences, it is necessary to sum over all possible amino acid elements of $\mathcal{S}$ that could have occurred. Therefore, the probability of observing the amino acid residues at

site n in Figure 2.2 becomes

$$f(x_n;\boldsymbol{\theta}) = \sum_{k_{\mathtt{A0}}}\sum_{k_{\mathtt{A1}}}\sum_{k_{\mathtt{A2}}} \pi^{(k_{\mathtt{A0}})} p_{k_{\mathtt{A0}}k_{\mathtt{A1}}}(t_1) p_{k_{\mathtt{A0}}k_{\mathtt{A2}}}(t_2) p_{k_{\mathtt{A1}}\mathtt{M}}(t_3) p_{k_{\mathtt{A1}}\mathtt{N}}(t_4) p_{k_{\mathtt{A2}}\mathtt{M}}(t_5) p_{k_{\mathtt{A2}}\mathtt{N}}(t_6), \tag{2.7}$$

where $k_{\mathtt{A0}}, k_{\mathtt{A1}}, k_{\mathtt{A2}} \in \mathcal{S}$. Given a substitution model that is reversible, Equation (2.7) simplifies to

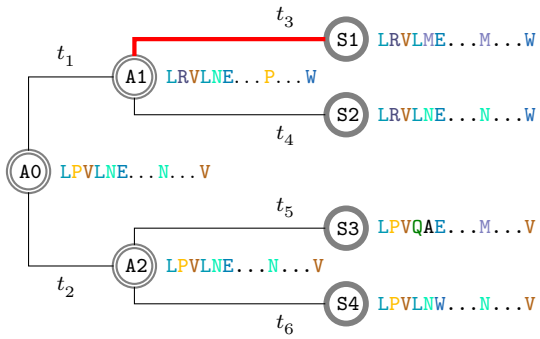
$$f(x_n;\boldsymbol{\theta}) = \sum_{k_{\mathtt{A1}}}\sum_{k_{\mathtt{A2}}} \pi^{(k_{\mathtt{A1}})} p_{k_{\mathtt{A1}}k_{\mathtt{A2}}}(t_{1.2}) p_{k_{\mathtt{A1}}\mathtt{M}}(t_3) p_{k_{\mathtt{A1}}\mathtt{N}}(t_4) p_{k_{\mathtt{A2}}\mathtt{M}}(t_5) p_{k_{\mathtt{A2}}\mathtt{N}}(t_6), \tag{2.8}$$

where $t_{1.2} = t_1 + t_2$. Felsenstein's (1973; 1981) pruning algorithm reduces the cost of summing over all ancestral states. Applying that technique to equations (2.7) and (2.8) respectively produced

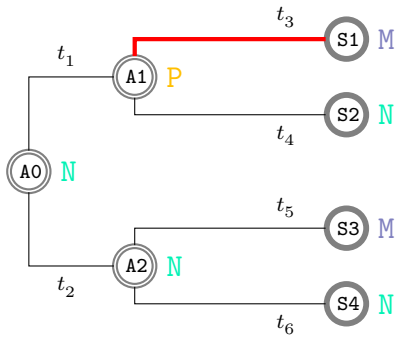
$$f(x_n;\boldsymbol{\theta}) = \sum_{k_{\mathtt{A0}}} \left(\pi^{(k_{\mathtt{A0}})} \sum_{k_{\mathtt{A1}}} \left(p_{k_{\mathtt{A0}}k_{\mathtt{A1}}}(t_1) p_{k_{\mathtt{A1}}\mathtt{M}}(t_3) p_{k_{\mathtt{A1}}\mathtt{N}}(t_4) \sum_{k_{\mathtt{A2}}} \left(p_{k_{\mathtt{A0}}k_{\mathtt{A2}}}(t_2) p_{k_{\mathtt{A2}}\mathtt{M}}(t_5) p_{k_{\mathtt{A2}}\mathtt{N}}(t_6) \right) \right) \right), \tag{2.9}$$

	1	4	7	...	n	...	N
S1	L	R	V	L	M	E	D	N	...	M	...	W
S2	L	R	V	L	N	E	H	N	...	N	...	W
S3	L	P	V	Q	A	E	H	E	...	M	...	V
S4	L	P	V	L	N	W	H	E	...	N	...	V

(a) Sequence alignment.



(b) Full phylogeny.



(c) Site n zoomed.

Figure 2.2: Alignment and phylogeny of four observed homologous protein coding sequences {S1, S2, S3, S4} and the unobserved ancestor sequences, {A0, A1, A2}. Divergence time t_r corresponds to the length (i.e. horizontal extension) of branch r. The sequence headings represent site indices.

for a non-reversible substitution model and

$$f(x_n;\boldsymbol{\theta}) = \sum_{k_{\mathtt{A1}}} \left(\pi^{(k_{\mathtt{A1}})} p_{k_{\mathtt{A1}}\mathtt{M}}(t_3) p_{k_{\mathtt{A1}}\mathtt{N}}(t_4) \sum_{k_{\mathtt{A2}}} \left(p_{k_{\mathtt{A1}}k_{\mathtt{A2}}}(t_{1.2}) p_{k_{\mathtt{A2}}\mathtt{M}}(t_5) p_{k_{\mathtt{A2}}\mathtt{N}}(t_6) \right) \right), \tag{2.10}$$

for a reversible substitution model. The likelihood for the full alignment is constructed by assuming independent sites, as specified in Equation (2.6). The substitution model parameters can then be estimated by likelihood maximisation. Modelling alignments with more than four sequences will only require a direct extension of the concepts in this subsection.

2.1.3 Types of amino acid models of evolution

Amino acid substitution models can be classified as mechanistic or empirical based on how the substitution rate matrix, $\mathbb{Q} = \mathbb{S}\Pi$, is constructed. Both types of models often assume that $\mathbb{S}$ is reversible. This research uses empirical amino acid models such as WAG (Whelan and Goldman, 2001), JTT (Jones et al., 1992) and HIV-B (Nickle et al., 2007).

JTT and WAG were generated from collection of protein sequences from a few (e.g. WAG used 182) families of mammalian genomes. HIV-B is one of two empirical substitution matrices constructed from a collection of HIV-1 alignments sampled from different viral genes. The second matrix, HIV-W, was designed to account for intra-host evolution and it was constructed from a combination of alignments that were each generated from a specific patient. HIV-B was generated from combination of alignments that were each sampled from separate patients. In practice, the equilibrium frequency component, Π, of empirical models are replaced by estimates from the data being analysed as this leads to more accurate inferences (Cao et al., 1994). When this is the case, the empirical model is often suffixed with "-F" e.g. JTT-F.

Figure 2.3 provides a visual illustration of the empirical models that are used in this research. The area of the bubbles in the $\mathbb{S}$ matrix plots are proportional to the corresponding exchangeabilities.

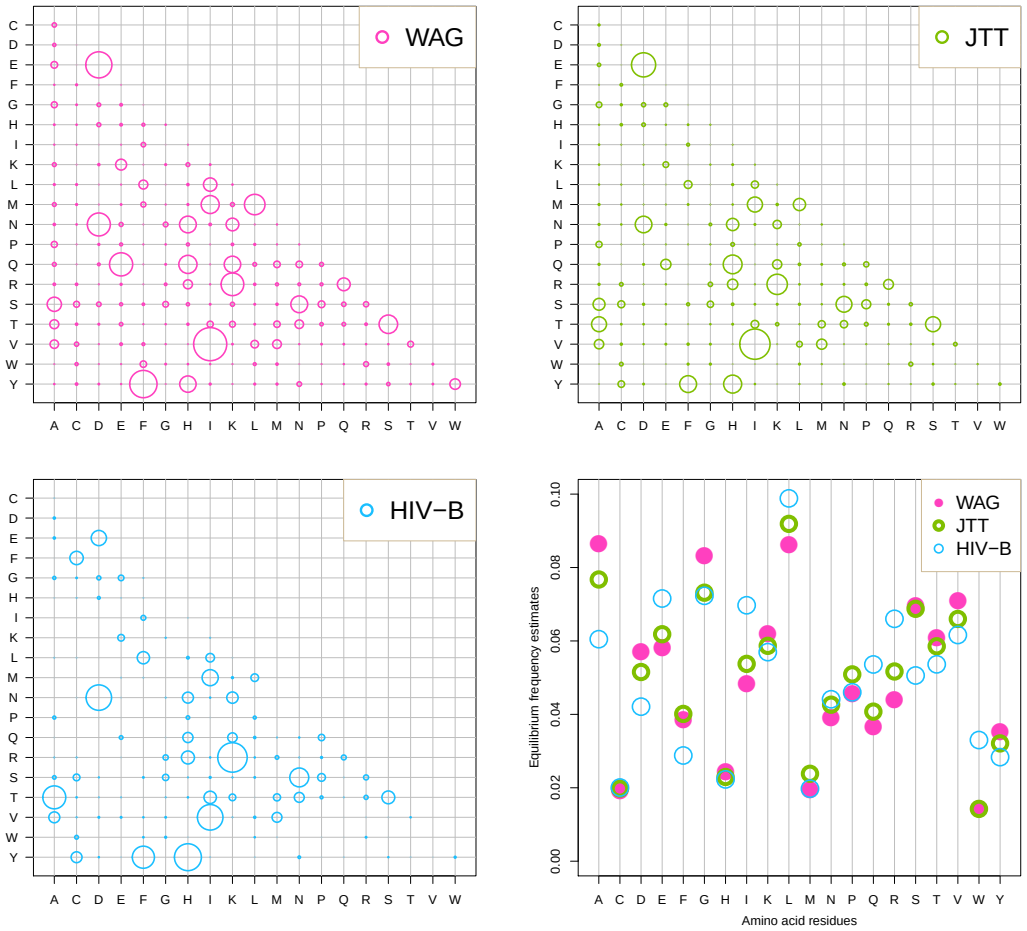


Figure 2.3: Illustration of WAG (Whelan and Goldman, 2001), JTT (Jones et al., 1992) and, HIV-B (Nickle et al., 2007) empirical instantaneous substitution matrices, $\mathbb{S}$, arranged from left-to-right then top-to-bottom respectively. The bottom-right plot shows the associated empirical equilibrium frequencies. The area of the bubbles on the $\mathbb{S}$ matrix plots is proportional to the corresponding exchangeability, s_{ij}.

2.2 Rate heterogeneity among sites

The explanations thus far have assumed that an entire amino acid sequence alignment can be modelled by a single 20×20 instantaneous substitution matrix $\mathbb{Q}$. This assumption is unrealistic because sequence positions are subject to different evolutionary pressures depending on their functional and structural roles in the protein (Goldman et al., 1998). Failure to account for non-homogeneous substitution rates among sites may cause biased infer-

ences (Jia et al., 2014). Various methods for incorporating rate heterogeneity among sites into models of amino acid evolution are described in this section.

Claiming that there exists rate heterogeneity among sites is equivalent to claiming that the sites in the alignment are governed by at least two separate $\mathbb{Q}$ matrices. It will be inefficient to attempt to estimate a separate $\mathbb{Q}$ for each site because, even when reversibility is assumed, there will be need to estimate 190 parameters per site for an UNREST amino acid model. This is tedious and unreliable. A common and efficient approach is to assume that the full alignment is characterised by a baseline matrix of exchangeabilities, $\mathbb{S} = \{s_{ij}\}$, such that each site differs only by a rate scaling parameter, $\mathcal{A}_n$. Each site n can then be analysed with site-specific substitution model, $\mathcal{A}_n\mathbb{S}$ and among site heterogeneity analyses is consequently reduced to inference about $\mathcal{A} = \{\mathcal{A}_n\}$ for $n = 1, 2, \ldots, N$.

Nielsen and Yang (1998) proposed inferring $\mathcal{A}_n$ for each site n in an alignment as the maximum likelihood estimate using the data available at that site, x_n. This was called the fixed effects approach. With this approach, the site-specific rates in an alignment of length N are usually scaled such that $\sum_{n=1}^{N} \mathcal{A}_n = 1$. The branch lengths may then still be interpreted as the expected number of substitutions. Except for the scaled $\mathbb{Q}$ and an expanded $\boldsymbol{\theta} = \{\mathbb{S}, \Pi, \tau, \mathbf{t}, \mathcal{A}\}$, computation of the probability of observing the residues at a particular site, $f(x_n; \boldsymbol{\theta}, \mathcal{A}_n)$ is as described in equations (2.9) and (2.10). Then, with the assumption of independent sites, the likelihood of the observed alignment under fixed effects approach can be calculated as

$$\mathcal{L}(\boldsymbol{\theta}, \mathcal{A}; X) = \prod_{n=1}^{N} f(x_n; \boldsymbol{\theta}, \mathcal{A}_n).$$

An advantage of the fixed effects approach is that $\mathcal{A}_n$ is free to take on any value at each site. A disadvantage of this approach, however, is that it introduces n new unknown parameters to the model. Also, alignment-wide information that might help make the estimate of $\mathcal{A}_n$ more reliable is ignored.

An alternate method to account for rate variation between sites was first used in the PAML package by Yang (1997). This method is commonly referred to as the random effects approach. It circumvents the two disadvantages of the fixed effects technique that were highlighted above. With the random effects approach, $\mathcal{A}$ is treated as a random variable drawn at each site from a probability distribution. A gamma probability distribution was assumed for the following explanation but the idea directly applies to other distributions. If $\mathcal{A}$ is assumed to be a continuous $\texttt{Gamma}(\alpha, \lambda)$, distributed random variable,

then the corresponding probability distribution can be specified as

$$g(\mathcal{A}) = \frac{\lambda^{\alpha}\mathcal{A}^{\alpha-1}\exp(\mathcal{A}\lambda)}{\Gamma(\alpha)}; \qquad \mathcal{A} > 0,$$

where α and λ respectively represent the shape and scale parameters of the distribution and the gamma function $\Gamma(\alpha)$ is given by

$$\Gamma(\alpha) = \int_0^{\infty} u^{\alpha-1}\exp(-u)\,\mathrm{d}u.$$

The mean and variance of $\mathcal{A} \sim \texttt{Gamma}(\alpha, \lambda)$ may be specified as

$$\mathrm{E}[\mathcal{A}] = \frac{\alpha}{\lambda}, \qquad \text{and} \qquad \mathrm{V}[\mathcal{A}] = \frac{\alpha}{\lambda^2},$$

respectively. The shape and scale parameters, α and λ, of the gamma distribution are usually set equal to each other such that $\mathcal{A} \sim \texttt{Gamma}(\alpha, \alpha)$ so that $\mathrm{E}[\mathcal{A}] = 1$.

Then, the likelihood of an observed alignment, X, for the random effects approach, under the assumption of independent sites, may be computed as

$$\mathcal{L}\big(\boldsymbol{\theta}, \mathcal{A}, \alpha; X\big) = \prod_{n=1}^{N}\int_0^{\infty} f(x_n; \boldsymbol{\theta}, \mathcal{A}_n)g(\mathcal{A}_n|\alpha)\,\mathrm{d}\mathcal{A}_n.$$

This likelihood is analytically intractable due to the integral component. As a result, the continuous gamma distribution is usually discretised into K categories and this is referred to as the discrete-gamma approach. In order to ensure a tractable likelihood function, K is often restricted to between 4 and 10 categories. The rate multiplier for each category, $\mathbb{A}^{(k)} \in \{\mathbb{A}^{(1)}, \ldots, \mathbb{A}^{(K)}\}$, is usually estimated as a function of α, as the mean or median of the corresponding discrete portion of the gamma distribution. Let $p_k = \Pr\big(\mathcal{A}_n = \mathbb{A}^{(k)}\big)$. The discrete-gamma approach uses $p_k = 1/K$, which corresponds to a valid (discrete uniform) probability distribution since

$$\sum_{k=1}^{K} p_k = 1. \tag{2.11}$$

If x_n is assumed to be independent then the discrete-gamma likelihood is given as

$$\mathcal{L}\big(\boldsymbol{\theta}, \mathcal{A}, \mathbf{p}; X\big) = \prod_{n=1}^{N}\sum_{k=1}^{K} f(x_n; \boldsymbol{\theta}, \mathbb{A}^{(k)})p_k,$$

where $\mathbf{p} = \{p_1, \ldots, p_K\}$. The discrete-gamma approach has the disadvantage that it imposes a distribution on $\mathcal{A}$. It is not always true that $\mathcal{A} \sim \texttt{Gamma}(\alpha, \alpha)$. To address this problem, Sorhannus and Kosakovsky Pond (2006) suggested a general discrete distribution (GDD) approach. Under GDD, the discrete rate categories, $\{\mathbb{A}^{(k)}\}$ and their associated probabilities, $\{p_k\}$, are estimated without assuming any underlying distribution. Given the constraint in Equation (2.11), it implies that GDD adds $2K - 1$ new parameters to the model. As a result, to avoid over-parameterisation, like the discrete-gamma, GDD restricts K to between 4 and 10 categories.

Murrell et al. (2013) showed that the restriction imposed on K by GDD leads to biased inference when the alignment being analysed is highly variable. Consequently, they proposed a fast unconstrained Bayesian approximation (FUBAR) method. With FUBAR, discrete rate parameters, $\{\mathbb{A}^{(k)}\}$, are not estimated, they are specified and the associated probabilities, $\{p_k\}$, are drawn from a $\texttt{Dirichlet}(\boldsymbol{\alpha})$ distribution, where $\boldsymbol{\alpha} = \{\alpha_k\}$. See Appendix A for an explanation of the Dirichlet distribution. FUBAR was shown by its authors to lead to more reliable inferences. As a result, this research adopts the FUBAR approach and its implementation is explored in the next chapter.

2.3 Rate heterogeneity across branches

Foster (2004) showed that branches in a phylogeny of amino acid sequences exhibit heterogeneous substitution rates that may compromise phylogenetic analyses if ignored. Models of evolution that accommodate rate variation across branches are termed episodic. In a highly cited article, Yang (1998) implemented several episodic models allowing for varying degrees of heterogeneity. The simplest of the implemented episodic models uses a rate parameter specified as, $\mathcal{B} > 0$. The saturated model hypothesises a separate $\mathcal{B} > 0$ for each branch.

The "single-rate" episodic model requires pre-classification of the phylogeny into two groups of foreground and background branches. The entire alignment is first assumed to be governed by a baseline matrix of exchangeabilities, $\mathbb{S} = \{s_{ij}\}$. This matrix is then scaled along the foreground branches such that the foreground matrix of exchangeabilites, $\mathbb{S}^{(F)} = \mathcal{B}\mathbb{S}$, while the background analogue is $\mathbb{S}^{(B)} = \mathbb{S}$. Episodic model analysis consequently becomes a problem of drawing inference about $\mathcal{B}$, which may then be treated as a rate category variable.

Zhang (2004) criticised the pre-requisite classification of branches based on an argument that such information is not available in some cases. Kosakovsky Pond and Frost (2005a) have since proposed a genetic algorithm approach for deciding on branch classification and Anisimova and Yang (2007) suggested benchmark sequential tests.

This research will adopt a "single-rate" approach since external information on the foreground and background branch classification is available in the applications considered here. This assumption is satisfied for analyses of sequences obtained from infected subjects before and after drug administration, for example.

2.4 Rate heterogeneity among sites and across branches

Substitution rates vary across sites and branches simultaneously and models that do not account for both are not accurate (Yang and Nielsen, 2002). The first attempt to simultaneously model rate heterogeneity among sites and across branches was the "branch-site" method proposed by Yang and Nielsen (2002). This model basically multiplied $\mathbb{S}^{(B)}$ and $\mathbb{S}^{(F)}$ by a site-specific rate multiplier, $\mathcal{A}_n$. The resulting branch-site model then had a background instantaneous substitution model of the form $\mathcal{A}\mathbb{S}^{(B)}$ and a foreground analogue given by $\mathcal{A}\mathbb{S}^{(F)}$.

Several extensions to the original model have since been proposed. Blanquart and Lartillot (2008) suggested a category mixture and non-stationary break point (CAT-BP) model that varies the Π component of $\mathbb{Q} = \mathbb{S}\Pi$ among sites and across branches for empirical amino acid models. Kosakovsky Pond et al. (2011) efficiently modelled site-and-branch rate variation under the random effects framework. Murrell et al. (2012) suggested a mixed effects model of evolution (MEME) where among site rate variation is modelled as fixed effects, while across branch rate variation is modelled as random effects.

2.4.1 Episodic directional selection

When randomly occurring mutations improve the ability of the organism to survive and reproduce, the mutants are said to be subject to positive natural selection. Positive selection can be classified as either diversifying or

directional in nature. In diversifying positive selection, substitutions toward any amino acid residue are favoured. Directional positive selection accelerates substitution toward only a small set of target residues. This research deals with directional selection. Many researchers have developed methods for detecting diversifying selection and the tutorial by Kosakovsky Pond et al. (2009) is a good starting point for the interested reader.

In their ground-breaking work, Murrell et al. (2012a) analysed the evolution of mutations conferring drug resistance in HIV-1 with an episodic directional selection model (EDEPS) based on datasets where a subset of the host population had undergone drug therapy. Relative to previous models that were either episodic or directional, they found that their approach provided a better description of the mechanism governing drug resistance.

Rate heterogeneity was modelled in EDEPS by closely following the method implemented in Kosakovsky Pond et al. (2008). As a result, site-specific rate multiplier (referred to as $\mathcal{A}_n$ here) was accounted for using the random effects approach with four discrete rate categories. Evolution of the set of target amino acid residues was modelled with a bias parameter. The bias parameter is referred to as $\mathcal{B}_n$ in this dissertation. Incorporation of the bias parameter was such that the phylogeny was first classified into sets of foreground and background branches. Then, along the foreground branches, substitution towards a set of target residues was accelerated while substitution away from the residues was hindered. Along the background branches, the bias parameter has a neutral effect. Both $\mathcal{A}_n$ and $\mathcal{B}_n$ are allowed to vary across sites. However, $\mathcal{B}_n$ only affects substitution rate when a target amino acid is involved, while $\mathcal{A}_n$ has nothing to do with the residues involved in the substitution.

The context for most of the models in the works discussed in this chapter were mostly defined in terms of codon. Contrarily, the model proposed in this dissertation is defined in terms of amino acids. This research uses techniques from FUBAR (Murrell et al., 2013) to implement a more robust extension of EDEPS where the constraint on the number of discrete categories used to model $\mathcal{A}$ and $\mathcal{B}$ is relaxed. Efficiency was improved by adopting inference techniques for the latent Dirichlet allocation model. The extended model was validated and compared by analysing the same empirical HIV-1 data sets as EDEPS. Sensitivity analyses were carried out using data sets that were simulated and analysed by closely following the methodologies of FUBAR, EDEPS and Kosakovsky Pond et al. (2008). Explanations of how these extensions and applications were achieved are presented in the next two chapters.

Chapter 3

Methodology

Murrell et al. (2012a) proposed a model for episodic directional evolution of protein sequences (EDEPS). The main interest was to detect sites with accelerated substitutions toward a set of amino acid residues. Rate heterogeneity among sites and across branches was accommodated.

In EDEPS, rate heterogeneity was modelled with four discrete random effect categories. It was assumed that a phylogeny could be partitioned into sets of foreground and background branches. Site-to-site rate variation was modelled with two parameters that were referred to in this dissertation as $\mathcal{A}_n$ and $\mathcal{B}_n$. The site-specific bias parameter, $\mathcal{B}_n$, was such that substitution towards a set of target residues, $\mathbf{y}_n$, was accelerated, while substitutions away from residues in the set were hindered. Unlike the bias parameter, the site-specific rate multiplier, $\mathcal{A}_n$, is independent of the amino acids involved in the substitution.

The episodic directional evolution modelling and inference approaches proposed in this research are similar to EDEPS and are described in this chapter. The adopted evolutionary model is described in the first section. The proposed inference algorithms were subsequently explained in the second section.

3.1 Model of episodic directional selection

The model of episodic directional evolution described here is defined for a phylogeny that can be classified into sets of foreground and background branches. Amino acid evolution along the background branches is described

by the substitution model $\mathbb{Q}^{(nB)} = \{q_{ij}^{nB}\}$ such that

$$q_{ij}^{(nB)} = \begin{cases} s_{ij} \times \pi^{(j)} \times \mathcal{A}_n & \text{for } i \neq j \\ -\sum_{j \neq i} q_{ij}^{(nB)} & \text{for } i = j, \end{cases}$$

while the foreground analogue is described by $\mathbb{Q}^{(nF)} = \{q_{ij}^{nF}\}$ such that

$$q_{ij}^{(nF)} = \begin{cases} s_{ij} \times \pi^{(j)} \times \mathcal{A}_n & \text{for } i,j \notin \mathbf{y}_n \text{ or } i,j \in \mathbf{y}_n,\ i \neq j \\ s_{ij} \times \pi^{(j)} \times \mathcal{A}_n \times t(\mathcal{B}_n)_+ & \text{for } i \notin \mathbf{y}_n,\ j \in \mathbf{y}_n \\ s_{ij} \times \pi^{(j)} \times \mathcal{A}_n \times t(\mathcal{B}_n)_- & \text{for } i \in \mathbf{y}_n,\ j \notin \mathbf{y}_n \\ -\sum_{j \neq i} q_{ij}^{(nF)} & \text{for } i = j, \end{cases}$$

where $n = 1, 2, \ldots, N$ represents the site index and $i, j \in \{\mathtt{A}, \mathtt{C}, \ldots, \mathtt{Y}\}$ refer to amino acid residues. The s_{ij} notation signifies the corresponding element of a pre-specified empirical matrix of exchangeabilities, $\mathbb{S}$. The scalar $\pi^{(j)}$ denotes the equilibrium frequency of amino acid j and was estimated from the sequence alignment as the observed proportion of each amino acid. Estimation of $\Pi = \{\pi^{(j)}\}$ from the alignment is conventional and has been shown, although in the context of codon models, not to adversely affect inferences (Kosakovsky Pond et al., 2010). The transformation functions $t(\star)_+$ and $t(\star)_-$ were adapted from Lacerda et al. (2010), such that

$$t(\mathcal{B}_n)_+ = \frac{\mathcal{B}_n}{1 - \exp(-\mathcal{B}_n)}, \qquad t(\mathcal{B}_n)_- = \frac{\mathcal{B}_n}{\exp(\mathcal{B}_n) - 1}.$$

Unlike in EDEPS, where bias parameter values were restricted to be at least one due to the use of a reciprocal function, this transformation only restricts the values to be non-negative.

The heterogeneous rates $\mathcal{A}_n$ and $\mathcal{B}_n$ were modelled as recommended by Murrell et al. (2013). They were pre-specified using the discretisation function

$$s(K, \psi, \eta) = \{s_1, s_2\},$$

with

$$\begin{aligned} s_1 &= \left\{ \frac{0}{\kappa}, \frac{1}{\kappa}, \ldots, \frac{\kappa}{\kappa} \right\} \\ s_2 &= \left\{ 1 + \delta^3, 1 + (2\delta)^3, \ldots, 1 + [(K - \kappa - 1)\delta]^3 \right\}, \end{aligned}$$

where

$$\begin{aligned} \kappa &= \lfloor \psi K \rfloor \\ \delta &= \frac{\sqrt[3]{\eta - 1}}{K - \kappa - 1}. \end{aligned}$$

The notation $\lfloor \# \rfloor$ refers to the maximum integer that is not greater than #. The number of rate categories is given by K. The scalar ψ gives the proportion of the rate values that should be less than one and η specifies the maximum value desired. Murrell et al. (2013) performed some experiments and suggested values of $K = 20$, $\psi = 0.70$ and $\eta = 50$. Let the pre-specified values for the site-specific rate multiplier be denoted by $\mathbb{A}$ such that the inferred rate for the nth site, $\hat{\mathcal{A}}_n$, is an element of $\mathbb{A}$. If $\mathbb{A} = s(20, 0.70, 50)$ then

$$\begin{aligned}\mathbb{A} &= \{\mathbb{A}^{(1)}, \mathbb{A}^{(2)}, \ldots, \mathbb{A}^{(20)}\} \\ &= \{\, 0.00,\ 0.07,\ 0.14,\ 0.21,\ 0.29,\ 0.36,\ 0.43,\ 0.50,\ 0.57,\ 0.64, \\ &\qquad 0.71,\ 0.79,\ 0.86,\ 0.93,\ 1.00,\ 1.39,\ 4.14,\ 11.58,\ 26.09,\ 50.00\}.\end{aligned}$$

The set of pre-defined site-specific bias parameters, $\mathbb{B}$, was also generated with the discretisation function. Let $K_{\mathbb{A}}$ represent the number of elements of $\mathbb{A}$ and let $K_{\mathbb{B}}$ denote the number of elements of $\mathbb{B}$. Then, the sample space of rate categories, $\mathbb{Z}$, with respect to the presented model of evolution was defined as a set of $K = K_{\mathbb{A}} \times K_{\mathbb{B}}$ elements as follows

$$\begin{aligned}\mathbb{Z} &= \{\mathbb{Z}^{(1)}, \mathbb{Z}^{(2)}, \ldots, \mathbb{Z}^{(k)}, \ldots, \mathbb{Z}^{(K)}\} \\ &= \left\{\left(\mathbb{A}^{(1)}, \mathbb{B}^{(1)}\right), \left(\mathbb{A}^{(1)}, \mathbb{B}^{(2)}\right), \ldots, \left(\mathbb{A}^{(1)}, \mathbb{B}^{(K_{\mathbb{B}})}\right), \left(\mathbb{A}^{(2)}, \mathbb{B}^{(1)}\right), \ldots, \left(\mathbb{A}^{(K_{\mathbb{A}})}, \mathbb{B}^{(K_{\mathbb{B}})}\right)\right\}\end{aligned}$$

The problem addressed in this research concerns inferring the bias parameter, $\mathcal{B}_n$, for every amino acid site in an observed alignment. Incorporation of site-specific rate multiplier in the model is to avoid confounded results because sites have different roles in protein. Analogous problems have been dealt with in the field of statistical natural language processing using what are known as topic models.

The type of topic model considered here is known as the latent Dirichlet allocation [LDA] model (Blei et al., 2003). In this research, inference techniques founded on LDA principles were developed to enhance efficient episodic directional selection (EDS) analysis. LDA models are described and then linked to EDS in the next section. The statistical problem addressed in EDS analyses is formally presented in Section 3.3. The particular LDA models considered and the adaptation processes are discussed in sections 3.4 and 3.5. The notations used in the next section are such that they easily link EDS to LDA.

3.2 Latent Dirichlet allocation

In statistical natural language processing, a document contains words that depend on unobserved topics. Let $\mathbb{Z} = \{\mathbb{Z}^{(1)}, \mathbb{Z}^{(2)}, \ldots, \mathbb{Z}^{(K)}\}$ be the K-

dimensional set of unique and unknown topics exhibited by a particular document, X, where K is the number of topics. Also, let $\mathbf{z} = \{z_1, z_2, \ldots, z_n, \ldots, z_N\}$ represent the N-dimensional vector that contains the topic that gave rise to each of the N words in a document such that $z_n \in \mathbb{Z}$, for all $n = 1, 2, \ldots, N$. Then, the probability of observing the nth word, x_n, in the document can be represented as

$$\Pr(x_n) = \sum_{k=1}^{K} \Pr(x_n | z_n = \mathbb{Z}^{(k)}) \Pr(z_n = \mathbb{Z}^{(k)}). \tag{3.1}$$

The primary goal is to infer the set of unobserved topics, $\mathbf{z}$, that characterises the observed document. To achieve this goal, a generative model is proposed. Generative models suggest a structure for the unobserved variables that gave rise to the observed dataset. In topic modelling, a structure with finite set of topics, $\mathbb{Z}$, is posited. Then, to generate a document, a topic is first drawn from a distribution. Given the topic, a word is drawn from a vocabulary.

Let $\boldsymbol{\varepsilon} = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_K\}$ represent the discrete probability distribution of the K topics governing the N words in an observed document. It can be described as the low-dimensional representation of the observed document in the "topic-space" (Darling, 2011). Mathematically,

$$\Pr(z_n = \mathbb{Z}^{(k)} | \boldsymbol{\varepsilon}) = \varepsilon_k, \tag{3.2}$$

subject to

$$\sum_{k=1}^{K} \varepsilon_k = 1.$$

Assume that there exists a fixed number of words, W, in the vocabulary such that each topic can be represented as a discrete distribution over all of the W words. For consistency with what is to follow, further assume that $W = N$. Let $\beta_{k\bullet} = \{\beta_{k1}, \beta_{k2}, \ldots, \beta_{kN}\}$, where β_{kn} denotes the probability that the nth word was from the kth topic. That is

$$\Pr(x_n | z_n = \mathbb{Z}^{(k)}, \beta_{k\bullet}) = \beta_{kn}. \tag{3.3}$$

Maximum likelihood estimates can be obtained for ε_k from Equation (3.1) using the Expectation-Maximisation algorithm (Hofmann, 2001; Dempster et al., 1977). The topic for the nth word, z_n, may then be inferred as the one corresponding to the highest ε_k estimate. This approach often results in local maxima and it converges slowly (Blei et al., 2003). LDA circumvents

this shortcoming by postulating that $\boldsymbol{\varepsilon}$ and $\boldsymbol{\beta}$ are observations from separate Dirichlet probability distributions.

Only $\boldsymbol{\varepsilon}$ was assumed to be a Dirichlet random variable in the use of LDA technique for EDS purposes. The $K \times N$ matrix, $\boldsymbol{\beta}$, was assumed to be known. This assumption is plausible in the context of EDS when the continuous space of rate parameters is approximated using the fixed grid from FUBAR. Contrarily, assuming a continuous space for topics is less intuitive in the context of topic modelling. This highlight represents a primary difference between traditional LDA and the present work.

The graph in Figure 3.1 represents random variables as nodes (shaded nodes correspond to observed variables). Dependent variables are connected by directed arrows, while plates show repetition of a substructure. The illustrated model posits that to generate a document, X, which contains N words, $x_1, x_2, \ldots, x_n, \ldots, x_N$,

1. Randomly sample a topic distribution, $\boldsymbol{\varepsilon}$, from a $\texttt{Dirichlet}(\boldsymbol{\alpha})$ distribution. See Appendix A for description of the Dirichlet distribution.
2. For each word $x_n \in X$:
   (a.) Randomly choose a topic $\mathbb{Z}^{(k)} \in \mathbb{Z}$ using the discrete topic distribution, $\boldsymbol{\varepsilon}$, sampled in step 1. Call this topic z_n and save it in $\mathbf{z}$ accordingly.

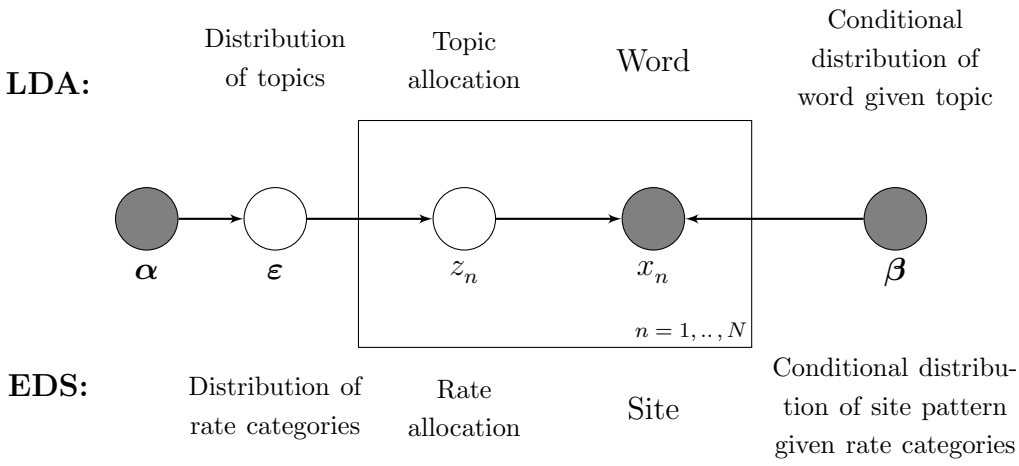


Figure 3.1: Graphical model illustration of an adjusted latent Dirichlet allocation (LDA) generative process and its relationship with episodic directional selection (EDS) analysis.

(b.) Randomly draw a word based on the N-dimensional probability vector, $\beta_{[z_n]\bullet} \subset \boldsymbol{\beta}$, corresponding to the topic chosen at step 2a, where $[z_n]$ is such that $[z_n] = k$ if $z_n = \mathbb{Z}^{(k)}$.

In topic modelling, multiple documents are needed in order to infer $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. Given that both parameters are assumed known in the EDS context, the graphical model in Figure 3.1 was restricted to a single gene/document. As a result, unlike traditional topic modelling, the illustration is a simpler two-level hierarchical model. The implied joint distribution can be expressed as

$$\Pr(\boldsymbol{\varepsilon}, \mathbf{z}, X|\boldsymbol{\alpha}, \boldsymbol{\beta}) = \Pr(\boldsymbol{\varepsilon}|\boldsymbol{\alpha}) \Pr(\mathbf{z}|\boldsymbol{\varepsilon}) \prod_{n=1}^{N} \Pr(x_n|z_n, \beta_{[z_n]\bullet}). \tag{3.4}$$

Let a document, X, be an alignment of protein sequences. Let words denote sites such that $x_1, x_2, \ldots, x_n, \ldots, x_N$ represent the N amino acid sites in X. Let topics represent rate categories, such that $z_n = \{\mathcal{A}_n, \mathcal{B}_n\}$ denotes the rate category governing the nth site. Let the sample space of rate categories be denoted by $\mathbb{Z} = \{\mathbb{Z}^{(k)}\}$. Replace the distribution over the N words in the vocabulary, $\boldsymbol{\beta}$, by the $K \times N$ matrix of conditional likelihoods such that

$$\beta_{kn} = \Pr\left(x_n|\boldsymbol{\theta}, \mathbb{Z}^{(k)}\right),$$

where $\boldsymbol{\theta} = \{\mathbb{S}, \Pi, \tau, \mathbf{t}\}$ is the set of evolution model parameters. It contains the baseline empirical matrix of exchangeabilities, $\mathbb{S}$, the equilibrium frequencies, Π, of amino acid residues, the evolution tree topology, τ, and a vector of branch lengths, $\mathbf{t}$. The correspondence between the LDA and the EDS models is illustrated in Figure 3.1.

3.3 Statistical problem

Combining equations (3.2) and (3.3) the joint distribution in equation (3.4) can be expressed as

$$\begin{aligned}
\Pr(\boldsymbol{\varepsilon}, \mathbf{z}, X|\boldsymbol{\alpha}, \boldsymbol{\beta}) &= \Pr(\boldsymbol{\varepsilon}|\boldsymbol{\alpha}) \Pr(\mathbf{z}|\boldsymbol{\varepsilon}) \prod_{n=1}^{N} \Pr\left(x_n|z_n, \beta_{[z_n]\bullet}\right) \\
&= \left(\frac{\Gamma\left(\sum_{k=1}^{K} \alpha_k\right)}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \prod_{k=1}^{K} \varepsilon_k^{\alpha_k - 1}\right) \left(\prod_{k=1}^{K} \varepsilon_k^{n_k}\right) \left(\prod_{k=1}^{K} \prod_{n=1}^{N} \beta_{kn}\right), \\
&= \frac{\Gamma\left(\sum_{k=1}^{K} \alpha_k\right)}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \prod_{k=1}^{K} \varepsilon_k^{\alpha_k - 1 + n_k} \left(\prod_{n=1}^{N} \beta_{kn}\right) \qquad (3.5)
\end{aligned}$$

where $[z_n] = k$ if $z_n = \mathbb{Z}^{(k)}$ and $n_k = \sum_{n=1}^{N} \mathbf{1}(z_n = \mathbb{Z}^{(k)})$ is the number of sites allocated to rate category $\mathbb{Z}^{(k)}$.

Exact inference about the rate category that gave rise to each site in an alignment requires employing the Bayesian idea of a posterior distribution. The joint posterior distribution of $\mathbf{z}$ and $\boldsymbol{\varepsilon}$ is given by

$$\Pr(\boldsymbol{\varepsilon}, \mathbf{z}|X, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{\Pr(\boldsymbol{\varepsilon}, \mathbf{z}, X|\boldsymbol{\alpha}, \boldsymbol{\beta})}{\int_{\boldsymbol{\varepsilon}} \sum_{\mathbf{z}} \Pr(\boldsymbol{\varepsilon}, \mathbf{z}, X|\boldsymbol{\alpha}, \boldsymbol{\beta}) \, \mathrm{d}\boldsymbol{\varepsilon}}. \qquad (3.6)$$

This equation is intractable due to the normalisation constant in the denominator, which depends on the latent variables $\mathbf{z}$ and $\boldsymbol{\varepsilon}$. As a result, the equation was approximated using stochastic and deterministic methods adapted from the LDA literature. The implemented approximation techniques include the collapsed Gibbs sampling (CGS) and the collapsed variational Bayes algorithms. These algorithms are discussed in the next sections.

The primary interest is to infer $\mathbf{z}$, therefore $\boldsymbol{\varepsilon}$ is a nuisance parameter that can be eliminated from Equation (3.6). Moreover, $\boldsymbol{\varepsilon}$ causes the CGS algorithm to converge slowly (Griffiths and Steyvers, 2004) and it makes the variational Bayes approximations inaccurate (Teh et al., 2007). Consequently, $\boldsymbol{\varepsilon}$ was

marginalised out from Equation (3.6) as follows

$$\begin{aligned}
\Pr(\mathbf{z}|X,\boldsymbol{\alpha},\boldsymbol{\beta}) &= \int_{\boldsymbol{\varepsilon}} \Pr(\boldsymbol{\varepsilon},\mathbf{z}|X,\boldsymbol{\alpha},\boldsymbol{\beta})\,\mathrm{d}\boldsymbol{\varepsilon} \\
&\propto \int_{\boldsymbol{\varepsilon}} \Pr(\boldsymbol{\varepsilon},\mathbf{z},X|\boldsymbol{\alpha},\boldsymbol{\beta})\,\mathrm{d}\boldsymbol{\varepsilon} \\
&\propto \int_{\boldsymbol{\varepsilon}} \frac{\Gamma\big(\sum_{k=1}^{K}\alpha_k\big)}{\prod_{k=1}^{K}\Gamma(\alpha_k)} \prod_{k=1}^{K} \varepsilon_k^{\alpha_k-1+n_k} \prod_{n=1}^{N} \beta_{[z_n]n}\,\mathrm{d}\boldsymbol{\varepsilon} \\
&\propto \frac{\prod_{k=1}^{K}\Gamma\big(\alpha_k+n_k\big)}{\Gamma\big(\sum_{k=1}^{K}\alpha_k+n_k\big)} \prod_{n=1}^{N} \beta_{[z_n]n}. \qquad (3.7)
\end{aligned}$$

Let $\mathbf{n} = \{n_1, n_2, \ldots, n_k, \ldots, n_K\}$, then the last line of the equation follows from the fact that

$$\prod_{k=1}^{K} \varepsilon_k^{\alpha_k-1+n_k} = \frac{\prod_{k=1}^{K}\Gamma\big(\alpha_k+n_k\big)}{\Gamma\big(\sum_{k=1}^{K}\alpha_k+n_k\big)} \times \texttt{Dirichlet}(\boldsymbol{\alpha}+\mathbf{n}).$$

3.4 Collapsed Gibbs sampling

The collapsed Gibbs sampling (CGS) algorithm was introduced by Griffiths and Steyvers (2004). It was developed by applying Gibbs sampling procedure to the marginalised posterior distribution expression in Equation (3.7). The Gibbs sampler is a stochastic algorithm from the Markov chain Monte Carlo family (Brooks et al., 2011). It aims to sample a chain of observations for a set of variables from an analytically intractable joint distribution using their conditional distributions, which are assumed to be known. Application of the CGS algorithm requires the conditional distribution of the rate category at each site, z_n, given the rates assigned to all other sites, $\mathbf{z}_{\neg n}$. This can be expressed as

$$\begin{aligned}
\Pr(z_n = \mathbb{Z}^{(k)}|\mathbf{z}_{\neg n},X,\boldsymbol{\alpha},\boldsymbol{\beta}) &= \frac{\Pr(z_n,\mathbf{z}_{\neg n},X|\boldsymbol{\alpha},\boldsymbol{\beta})}{\Pr(\mathbf{z}_{\neg n},X|\boldsymbol{\alpha},\boldsymbol{\beta})} \\
&\propto \Pr(\mathbf{z}|X,\boldsymbol{\alpha},\boldsymbol{\beta}).
\end{aligned}$$

Let $n_k = \sum_{n=1}^{N} \mathbf{1}(z_n = \mathbb{Z}^{(k)})$ represent the number of sites assigned to the kth rate category, $\mathbb{Z}^{(k)}$. Let $n_k^{\neg n} = n_k - \mathbf{1}(z_n = \mathbb{Z}^{(k)})$ denote the number of sites assigned to the kth rate category, excluding the nth site. Let $n_k^{=n} =$

$\mathbf{1}(z_n = \mathbb{Z}^{(k)})$ be a Bernoulli random variable that is equal to one if the nth site is assigned to the kth rate category and it equals zero otherwise. If the normalising terms and every component that does not depend on $\mathbb{Z}^{(k)}$ or x_n in Equation (3.7) are absorbed into the proportionality constant, then it follows that

$$\begin{aligned}
\Pr\big(z_n = \mathbb{Z}^{(k)} | \mathbf{z}_{\neg n}, X, \boldsymbol{\alpha}, \boldsymbol{\beta}\big) &\propto \beta_{[z_n]n} \prod_{j=1}^{K} \Gamma\big(\alpha_j + n_j\big) \\
&\propto \beta_{kn}\, \Gamma\big(\alpha_k + n_k\big) \prod_{j \neq k} \Gamma\big(\alpha_j + n_j^{\neg n}\big) \\
&\propto \beta_{kn}\, \Gamma\big(\alpha_k + n_k^{\neg n} + n_k^{=n}\big) \prod_{j \neq k} \Gamma\big(\alpha_j + n_j^{\neg n}\big) \\
&\propto \beta_{kn}\, \Gamma\big(\alpha_k + n_k^{\neg n} + 1\big) \prod_{j \neq k} \Gamma\big(\alpha_j + n_j^{\neg n}\big) \\
&\propto \beta_{kn}\, \big(\alpha_k + n_k^{\neg n}\big)\, \Gamma\big(\alpha_k + n_k^{\neg n}\big) \prod_{j \neq k} \Gamma\big(\alpha_j + n_j^{\neg n}\big) \\
&\propto \beta_{kn}\, \big(\alpha_k + n_k^{\neg n}\big) \prod_{j=1}^{K} \Gamma\big(\alpha_j + n_j^{\neg n}\big) \\
&\propto \beta_{kn}\, \big(\alpha_k + n_k^{\neg n}\big). \qquad (3.8)
\end{aligned}$$

Implementation of the CGS algorithm is subsequently straightforward. It requires three inputs. First, the value of the parameter, $\boldsymbol{\alpha}$, of the Dirichlet distribution imposed on the topic distribution. Second, the conditional likelihood matrix, $\boldsymbol{\beta}$. Third, the $K \times N$ initial estimates that each corresponds to the $\Pr\big(z_n = \mathbb{Z}^{(k)} | \mathbf{z}_{\neg n}, X, \boldsymbol{\alpha}, \boldsymbol{\beta}\big) = \gamma_{kn}^{(0)}$ such that $\boldsymbol{\gamma}^{(0)} = \{\gamma_{kn}^{(0)}\}$.

The matrix $\boldsymbol{\beta}$ was computed using the evolution model given in Section 3.1. For a tree topology, τ, maximum likelihood estimates of the branch lengths, $\hat{\mathbf{t}}$, were obtained using the background evolution model, $\mathbb{Q}^{(nB)} = \{q_{ij}^{(nB)}\}$, without rate variation. Following Murrell et al. (2013), a symmetric Dirichlet prior distribution with $\alpha_1 = \alpha_2 = \ldots = \alpha_K = 0.5$ was used. The initial matrix $\boldsymbol{\gamma}^{(0)}$ was derived from $\boldsymbol{\beta}$ (see Subsection 3.7).

Given these inputs, the CGS algorithm then adaptively iterates over Equation (3.8) for all k and all n. Every full iteration returns a $\boldsymbol{\gamma}$ sample. The CGS chain is guaranteed to converge to the true posterior distribution, $\Pr(\mathbf{z}|X, \boldsymbol{\alpha}, \boldsymbol{\beta})$. Efficient implementation of the CGS algorithm requires decisions about the time of convergence, N_{burn}, and the size of the sampling

window, N_{thin}. The purpose of the sampling window is to minimise the correlation among the samples of the Markov chain. Values of N_{burn} and N_{thin} were obtained from an exploratory simulation study that is presented in the next chapter.

Figure 3.2 presents a flow chart illustrating the CGS algorithm procedure. The scripts of the corresponding `HyPhy` and `R` code are presented in appendices B.1 and B.2 respectively. Every CGS execution was carried out with a stopping rule, N_{stop} =1,000 independent samples from the true posterior distribution. Estimates of the matrices of the posterior distribution of rate categories, $\hat{\boldsymbol{\gamma}}$'s, were subsequently obtained as the mean of the independent draws.

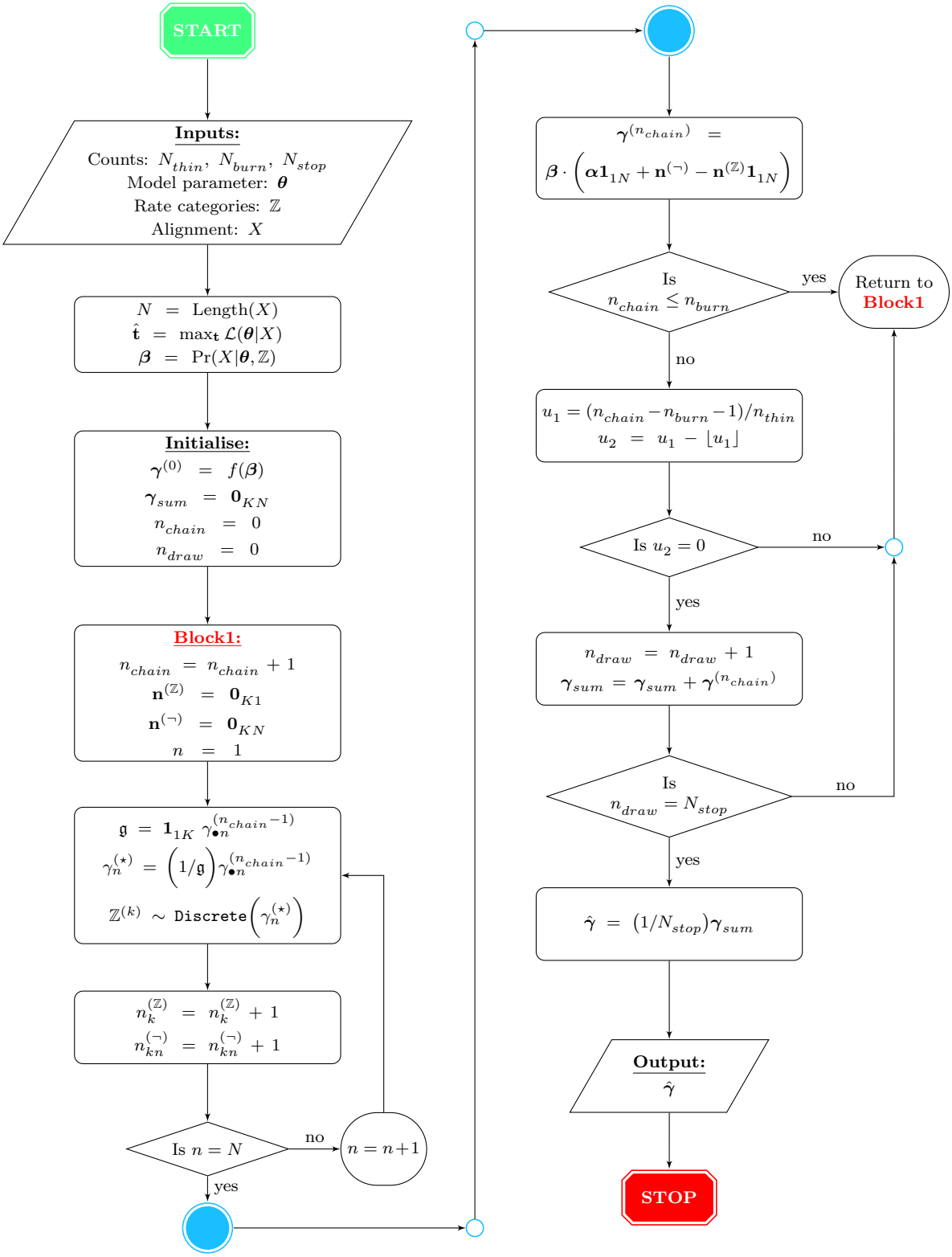


Figure 3.2: Illustration of the collapsed Gibbs sampling (CGS) algorithm. The notation $\mathbf{0}_{KN}$ represents a $K \times N$ matrix of zeros, $\mathbf{1}_{KN}$ is a $K \times N$ matrix of ones and $\gamma_{\bullet n}^{(.)} = \left(\gamma_{1n}^{(.)}, \gamma_{2n}^{(.)}, \ldots, \gamma_{Kn}^{(.)}\right)'$.

3.5 Collapsed variational Bayes

Variational Bayes (VB) methods provide deterministic approximations to intractable posterior distributions. The formulation of the problem is in terms of an optimisation functional. The goal is to identify a tractable distribution that closely approximates the true distribution. Dissimilarity between the distributions is measured by the Kullback-Leibler (KL) divergence (Kullback and Leibler, 1951). Given a family of approximate distributions, the ideal distribution is the one that minimises the KL divergence relative to the true distribution. The type of VB methods considered here are known as the factorised or the mean-field VB.

Factorised VB assumes that the joint intractable posterior distribution, $\Pr(\mathbf{z}|X, \boldsymbol{\alpha}, \boldsymbol{\beta})$, can be approximated by a tractable joint distribution $q(\mathbf{z})$. It is further assumed that $q(\mathbf{z})$ can be factorised into the product of the marginals, $q(z_1)$, $q(z_2)$, ..., $q(z_N)$ such that

$$q(\mathbf{z}) = \prod_{n=1}^{N} q(z_n). \tag{3.9}$$

In other words, the z_n are assumed to be independent. This assumption causes the factorised VB algorithms to produce less accurate results. Their qualities are therefore complementary to those of the stochastic sampling methods. There is however the advantage that they yield fast algorithms that can be easily applied to large datasets (Bishop, 2007).

Factorised VB algorithms seek the distribution, $q^{\star}(\mathbf{z})$, that minimises the KL divergence between $q(\mathbf{z})$ and $\Pr(\mathbf{z}|X, \boldsymbol{\alpha}, \boldsymbol{\beta})$ from the family of distributions that can be expressed in the form of Equation (3.9). Let the KL divergence

be represented by $\mathrm{KL}(\mathbf{q}||\mathbf{p})$. Consider the following probability expression

$$\begin{aligned}
\Pr(X) &= \frac{\Pr(\mathbf{z}, X|\boldsymbol{\alpha}, \boldsymbol{\beta})}{\Pr(\mathbf{z}|X, \boldsymbol{\alpha}, \boldsymbol{\beta})} \frac{q(\mathbf{z})}{q(\mathbf{z})} \\
\ln\Big(\Pr(X)\Big) &= \ln\left(\frac{\Pr(\mathbf{z}, X|\boldsymbol{\alpha}, \boldsymbol{\beta})}{q(\mathbf{z})}\right) - \ln\left(\frac{\Pr(\mathbf{z}|X, \boldsymbol{\alpha}, \boldsymbol{\beta})}{q(\mathbf{z})}\right) \\
\sum_{\mathbf{z}} q(\mathbf{z}) \ln\Big(\Pr(X)\Big) &= \sum_{\mathbf{z}} q(\mathbf{z}) \ln\left(\frac{\Pr(\mathbf{z}, X|\boldsymbol{\alpha}, \boldsymbol{\beta})}{q(\mathbf{z})}\right) - \sum_{\mathbf{z}} q(\mathbf{z}) \ln\left(\frac{\Pr(\mathbf{z}|X, \boldsymbol{\alpha}, \boldsymbol{\beta})}{q(\mathbf{z})}\right) \\
\ln\Big(\Pr(X)\Big) &= \sum_{\mathbf{z}} q(\mathbf{z}) \ln\left(\frac{\Pr(\mathbf{z}, X|\boldsymbol{\alpha}, \boldsymbol{\beta})}{q(\mathbf{z})}\right) - \sum_{\mathbf{z}} q(\mathbf{z}) \ln\left(\frac{\Pr(\mathbf{z}|X, \boldsymbol{\alpha}, \boldsymbol{\beta})}{q(\mathbf{z})}\right) \\
\ln\Big(\Pr(X)\Big) &= L(\mathbf{q}) + \mathrm{KL}(\mathbf{q}||\mathbf{p}), \qquad (3.10)
\end{aligned}$$

where

$$\begin{aligned}
\mathrm{KL}(\mathbf{q}||\mathbf{p}) &= -\sum_{\mathbf{z}} q(\mathbf{z}) \ln\left(\frac{\Pr(\mathbf{z}|X, \boldsymbol{\alpha}, \boldsymbol{\beta})}{q(\mathbf{z})}\right). \qquad (3.11) \\
L(\mathbf{q}) &= \sum_{\mathbf{z}} q(\mathbf{z}) \ln\left(\frac{\Pr(\mathbf{z}, X|\boldsymbol{\alpha}, \boldsymbol{\beta})}{q(\mathbf{z})}\right). \qquad (3.12)
\end{aligned}$$

Equation (3.12) represents the lower bound of the log-likelihood of the observed alignment, $\ln[\Pr(X)]$. Since $\ln[\Pr(X)]$ is a constant, minimising $\mathrm{KL}(\mathbf{q}||\mathbf{p})$ is equivalent to maximising the lower bound, $L(\mathbf{q})$, and the maximum of $L(\mathbf{q})$ occurs when the KL term is zero. From Equation (3.11), the KL divergence will be zero when $\Pr(\mathbf{z}|X, \boldsymbol{\alpha}, \boldsymbol{\beta}) = q(\mathbf{z})$.

An expression for the optimal approximate distribution, $q^{\star}(\mathbf{z})$, can therefore be obtained by maximising the lower bound in Equation (3.12) after replacing $q(\mathbf{z})$ with its factorised form from Equation (3.9). The procedure is as follows

$$\begin{aligned}
\mathrm{L}(\mathbf{q}) &= \sum_{\mathbf{z}} q(\mathbf{z}) \ln \frac{\Pr(\mathbf{z}, X|\boldsymbol{\alpha}, \boldsymbol{\beta})}{q(\mathbf{z})} \\
&= \mathrm{E}_{\mathbf{z}}\big[\ln \Pr(\mathbf{z}, X|\boldsymbol{\alpha}, \boldsymbol{\beta})\big] - \mathrm{E}_{\mathbf{z}}\big[\ln q(\mathbf{z})\big] \\
&= \mathrm{E}_{\mathbf{z}}\big[\ln\big\{\Pr(z_n|\mathbf{z}_{\neg n}, X, \boldsymbol{\alpha}, \boldsymbol{\beta}) \Pr(\mathbf{z}_{\neg n}|X, \boldsymbol{\alpha}, \boldsymbol{\beta})\big\}\big] - \sum_{n=1}^{N} \mathrm{E}_{z_n}\big[\ln q(z_n)\big].
\end{aligned}$$

Expressing the equation in terms of a single factor $q(z_n)$ results in

$$\begin{aligned} L(q_n) &= \mathrm{E}_{\mathbf{z}}\big[\ln \Pr(z_n|\mathbf{z}_{\neg n}, X, \boldsymbol{\alpha}, \boldsymbol{\beta})\big] - \mathrm{E}_{z_n}\big[\ln q(z_n)\big] + \text{constant} \\ &= \sum_{z_n} q(z_n)\, \mathrm{E}_{\mathbf{z}_{\neg n}}\big[\ln \Pr(z_n|\mathbf{z}_{\neg n}, X, \boldsymbol{\alpha}, \boldsymbol{\beta})\big] - \sum_{z_n} q(z_n) \ln q(z_n) + \text{constant} \\ &= \sum_{z_n} q(z_n)\, \ln \mathbb{E}_{\mathbf{z}_{\neg n}}\big[z_n\big] - \sum_{z_n} q(z_n) \ln q(z_n) + \text{constant} \\ &= -\Big\{ -\sum_{z_n} q(z_n)\, \ln\left(\frac{\mathbb{E}_{\mathbf{z}_{\neg n}}\big[z_n\big]}{q(z_n)}\right)\Big\} + \text{constant}, \end{aligned} \tag{3.13}$$

where

$$\mathbb{E}_{\mathbf{z}_{\neg n}}\big[z_n\big] = \exp\Big\{\mathrm{E}_{\mathbf{z}_{\neg n}}\big[\ln \Pr(z_n|\mathbf{z}_{\neg n}, X, \boldsymbol{\alpha}, \boldsymbol{\beta})\big]\Big\}.$$

Except for the constant term, Equation (3.13) represents the negative of the KL divergence between $\mathbb{E}_{\mathbf{z}_{\neg n}}\big[z_n\big]$ and $q(z_n)$ and the equation is maximised when $\mathbb{E}_{\mathbf{z}_{\neg n}}\big[z_n\big] = q(z_n)$. Therefore, $q^{\star}(z_n)$, can be expressed as

$$\begin{aligned} q^{\star}(z_n) &\propto \exp\big(\mathrm{E}_{\mathbf{z}_{\neg n}}\big[\ln \Pr(z_n|\mathbf{z}_{\neg n}, X, \boldsymbol{\alpha}, \boldsymbol{\beta})\big]\big) \\ &\propto \exp\big(\mathrm{E}_{\mathbf{z}_{\neg n}}\big[\ln\big\{\beta_{kn}\left(\alpha_k + n_k^{\neg n}\right)\big\}\big]\big) \\ &\propto \beta_{kn} \exp\big(\mathrm{E}_{\mathbf{z}_{\neg n}}\big[\ln\left(\alpha_k + n_k^{\neg n}\right)\big]\big). \end{aligned} \tag{3.14}$$

The second line of the equation follows from Equation (3.8). Equation (3.14) does not provide an explicit solution since it depends on $\mathrm{E}_{\mathbf{z}_{\neg n}}\big[\ln\left(\alpha_k + n_k^{\neg n}\right)\big]$. Exact estimation of this expectation is computationally tedious. Consequently, Teh et al. (2007) developed an accurate Gaussian approximation based on the second-order Taylor series expansion of $\ln(\alpha_k + n_k^{\neg n})$ about $\mathrm{E}_{z_n}[n_k^{\neg n}]$. This yielded

$$\mathrm{E}_{\mathbf{z}_{\neg n}}\big[\ln\left(\alpha_k + n_k^{\neg n}\right)\big] \approx \ln\left(\alpha_k + \mathrm{E}_{\mathbf{z}_{\neg n}}[n_k^{\neg n}]\right) - \frac{\mathrm{Var}_{\mathbf{z}_{\neg n}}[n_k^{\neg n}]}{2\big(\alpha_k + \mathrm{E}_{\mathbf{z}_{\neg n}}[n_k^{\neg n}]\big)^2}. \tag{3.15}$$

The value of $n_k^{\neg n}$ is a sum of $N-1$ random Bernoulli variables. Therefore, given a large number of sites, $n_k^{\neg n}$ can be approximated with a Gaussian distribution with

$$\begin{aligned} \mathrm{E}_{\mathbf{z}_{\neg n}}[n_k^{\neg n}] &= \sum_{j \neq n} \gamma_{kj} \\ \mathrm{Var}_{\mathbf{z}_{\neg n}}[n_k^{\neg n}] &= \sum_{j \neq n} \gamma_{kj}\big(1 - \gamma_{kj}\big). \end{aligned} \tag{3.16}$$

From equations (3.14), (3.15) and (3.16), the resulting element-wise update equation can be expressed as

$$\gamma_{kn} = \beta_{kn}\left(\alpha_k + \sum_{j\neq n}\gamma_{kj}\right)\exp\left\{-\frac{\sum_{j\neq n}\gamma_{kj}(1-\gamma_{kj})}{2\left(\alpha_k+\sum_{j\neq n}\gamma_{kj}\right)^2}\right\}. \tag{3.17}$$

This is referred to as the collapsed variational Bayes (CVB) update equation. Teh et al. (2007) performed some experiments and found that the Gaussian approximation works well even for small N.

The CVB algorithm, like the CGS, requires $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}^{(0)}$. These inputs were specified as for the CGS algorithm, following Murrell et al. (2013) (see Subsection 3.7). After initialisation, the algorithm then adaptively iterates over Equation (3.17) for all k and all n until convergence. Convergence is guaranteed because the update equation is a special type of the Expectation-Maximisation model (Massingham and Goldman, 2005).

Let $\boldsymbol{\gamma}^{(m)}$ represent the posterior probability matrix obtained after the mth iteration. The algorithm was designed to terminate if the maximum absolute difference between the elements of $\boldsymbol{\gamma}^{(m)}$ and $\boldsymbol{\gamma}^{(m-1)}$ (i.e. the error tolerance) is less than 10^{-20} or after the 1,000th iteration. The choice of 1,000 was from observation that the error tolerance at such point was often below 10^{-5}, while the choice of 10^{-20} was arbitrary. Figure 3.3 presents a flow chart illustrating the CVB algorithm.

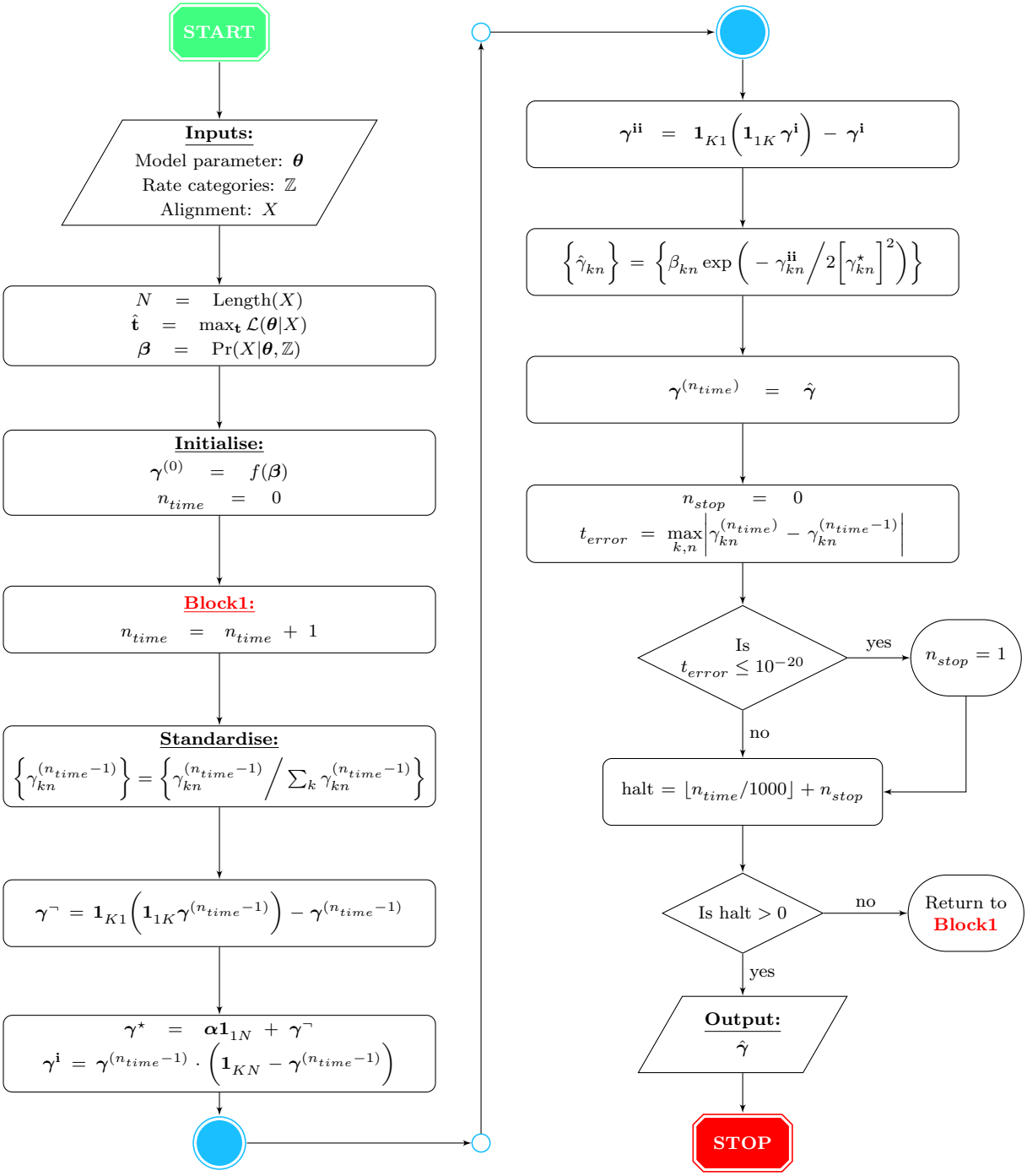


Figure 3.3: Flow chart illustration of the collapsed variational Bayes (CVB) algorithm. The notation $\mathbf{1}_{KN}$ denotes a $K \times N$ matrix of ones.

3.6 Zero-order collapsed variational Bayes

The zero-order collapsed variational Bayes algorithm (CVB0) was introduced by Asuncion et al. (2009). Its derivation differs from that of the CVB only by replacing the second-order Taylor expansion by its zero-order analogue. The resulting update function can be expressed as

$$\gamma_{kn} = \beta_{kn}\left(\alpha_k + \sum_{j \neq n} \gamma_{kj}\right) \tag{3.18}$$

The initial distribution matrix and convergence criteria used for implementing this algorithm was exactly as described for the CVB approach. The flow chart in Figure 3.4 presents the implementation procedure.

Various simulation studies carried out to compare the CGS, CVB and CVB0 algorithms are presented in the next chapter. The CGS algorithm served as the standard for evaluating the performances of the CVB and the CVB0 techniques. The initial posterior probability matrix, $\boldsymbol{\gamma}^{(0)}$, was identically generated for all the algorithms and the initialisation process is described in Subsection 3.7. Subsection 3.8 contains details of how inferences about the branch-specific rate and the set of targeted amino acid residues were drawn from the estimate, $\hat{\boldsymbol{\gamma}}$, obtained from implementing the three algorithms. The information in the following subsections are based on recommendations from Murrell et al. (2013).

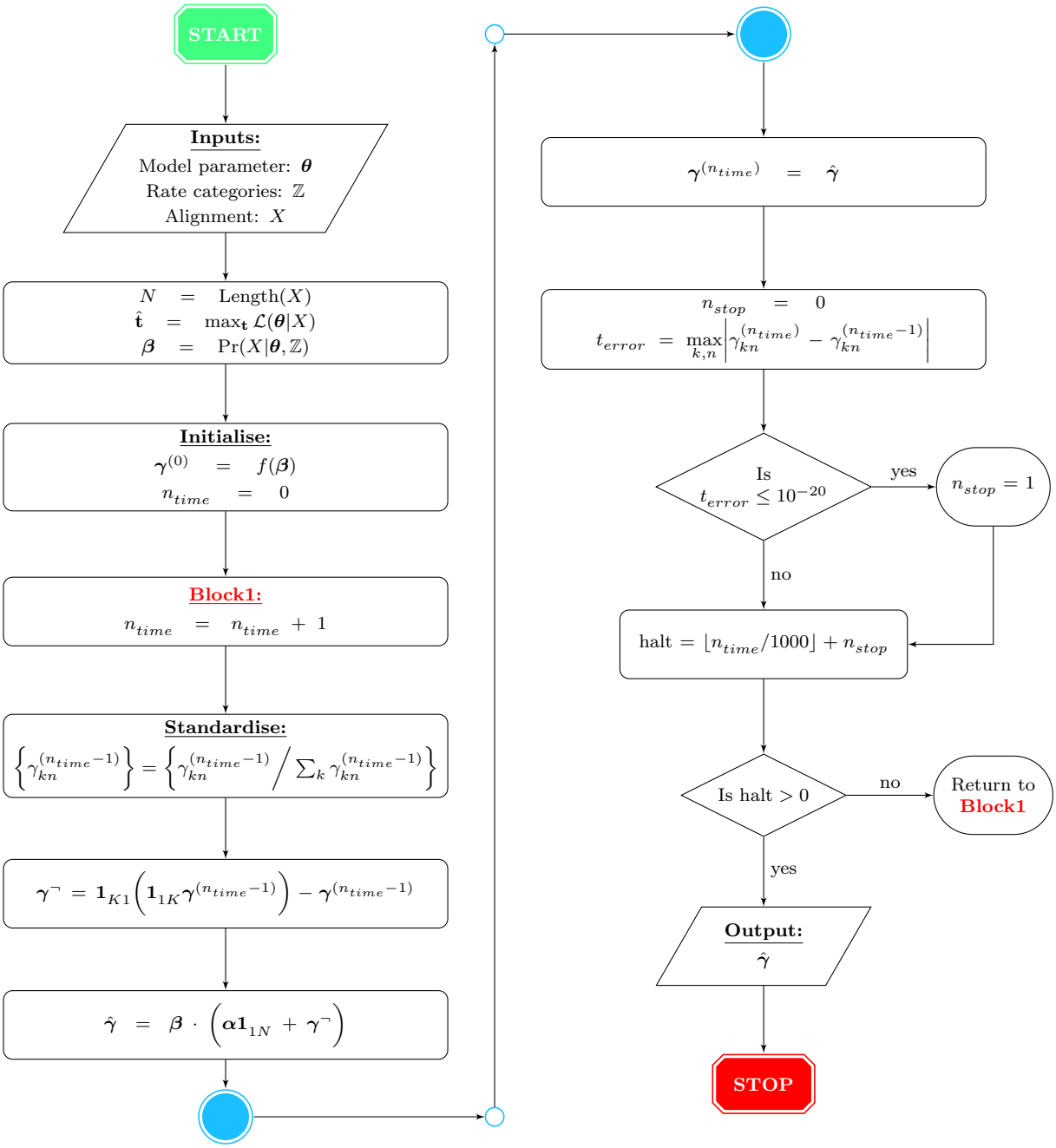


Figure 3.4: Diagrammatic representation of the zero-order collapsed variational Bayes (CVB0) algorithm, where $\mathbf{1}_{KN}$ is a $K \times N$ matrix of ones.

3.7 Initialisation

The initial posterior probability matrix, $\boldsymbol{\gamma}^{(0)} = \{\gamma_{kn}^{(0)}\}$, was computed as a function of the conditional likelihood matrix, $\boldsymbol{\beta} = \{\beta_{kn}\}$ as follows

$$\gamma_{kn}^{(0)} = \frac{\Lambda_{kn}}{\sum_{k=1}^{K} \Lambda_{kn}},$$

with

$$\Lambda_{kn} = \frac{\sum_{k=1}^{K} \beta_{kn}}{\sum_{n=1}^{N} \beta_{kn}} \times \mathtt{e}_0,$$

where $\mathtt{e}_0 \sim \mathtt{Uniform}(0.8, 1.2)$. If $\min(\Lambda) = 0$ then, Λ_{kn} was replaced with $\Lambda_{kn} + \mathtt{e}_1$, where $\mathtt{e}_1 \sim \mathtt{Uniform}(0.5, 1.0)$.

3.8 Bayes factors

After obtaining an estimate of the matrix of posterior probabilities, $\hat{\boldsymbol{\gamma}}$, the matrix was normalised such that for each site, the probabilities sum to one over the pre-defined rate categories. It was then left to decide on the most suitable rate assignment for each site. The main interest in this work is to infer the site-specific bias parameter, $\mathcal{B}_n$. Thus, the null, $\mathtt{H}_0$, and alternative, $\mathtt{H}_1$, hypotheses considered at each site n in the alignment are as follows

$\mathtt{H}_0$: The rate of evolution toward the defined set of target residues does not differ between background and foreground branches. That is, $\mathcal{B}_n = 0$.

$\mathtt{H}_1$: The rate of evolution toward the defined set of target residues along the foreground branches is greater than that along the background branches. That is, $\mathcal{B}_n > 0$.

The test statistic used is the Bayes factor (Kass and Raftery, 1995). The Bayes factor for each site in the alignment was computed as

$$\mathtt{BF}_n = \frac{\text{posterior odds ratio}}{\text{prior odds ratio}} = \frac{\sum_k (\hat{\gamma}_{kn} | \mathbb{B}^{(k)} > 0)}{\sum_k (\hat{\gamma}_{kn} | \mathbb{B}^{(k)} = 0)} \Big/ \frac{\sum_n \sum_k (\hat{\gamma}_{kn} | \mathbb{B}^{(k)} > 0)}{\sum_n \sum_k (\hat{\gamma}_{kn} | \mathbb{B}^{(k)} = 0)}.$$

Consequently the inferred bias parameter for the nth site, $\hat{\mathcal{B}}_n$, with respect to each element of the set of target residues was derived as follows

$$\hat{\mathcal{B}}_n \begin{cases} = 0 & \text{if } \texttt{BF}_n < 100 \\ \\ > 0 & \text{if } \texttt{BF}_n \geq 100 \end{cases}$$

The choice of 100 as the inference threshold was in accordance to the choice made by Murrell et al. (2013). However, in some unreported simulation studies carried out in the course of this research, it was observed that the inferences from the algorithms were not adversely affected by choices of thresholds between 10 and 100.

3.9 Computational resources

Software. All analyses were implemented with the Hypothesis Testing Using Phylogenies [HyPhy] (Kosakovsky Pond et al., 2005) software, version `2.220140721 beta(MPI) for Darwin on x86_64`. Some complementary analyses were performed in `R` (R Core Team, 2013), version `3.1.1 for Platform: x86_64- apple-darwin13.1.0 (64-bit)`. The `HyPhy` batch files and `R` scripts that contain the main functions are provided in the appendix.

Hardware. All of the code used was compiled on `MacBook OS X 10.9.4 Mavericks`. Implementation of the analyses were carried out on facilities supplied by the High Performance Computing team, Information and Communication Technology Service, University of Cape Town (`http://hpc.uct.ac.za`), and by the Center for AIDS Research, University of California, San Diego.

Chapter 4

Simulation study

Simulation studies were conducted to examine the relative performances of the collapsed Gibbs sampling [CGS] (Griffiths and Steyvers, 2004), the collapsed variational Bayes [CVB] (Teh et al., 2007), and the zero-order collapsed variational Bayes [CVB0] (Asuncion et al., 2009) algorithms. The analysis procedures and results are presented in this chapter. All analyses were implemented in both `HyPhy` (Kosakovsky Pond et al., 2005) and `R` (R Core Team, 2013). The implementations in `R` were for comparison purposes only, and the results agreed with those obtained from `HyPhy`. Therefore, only `HyPhy` results are reported. The `R` and `HyPhy` scripts are presented in the appendix with supporting information on how to use them.

Each simulation involved data generation and modelling, which required some parameter settings. The default specifications used are presented in the next section. The diagnostic analyses for the CGS algorithm are discussed in the subsequent section. In the third section, CGS, CVB, and CVB0, are compared and the most efficient algorithm is identified. The effects of varying some components of the default specifications are investigated in the concluding section.

4.1 Default simulation specifications

In this section, the default parameter settings used for the simulation and modelling of protein-coding sequences subjected to episodic directional selection (EDS) are presented. Most of the specifications were adopted directly from the FUBAR (Murrell et al., 2013) and the EDEPS (Murrell et al., 2012a) papers.

4.1.1 Default settings for alignment generation

The following settings were employed to simulate an alignment under EDS at selected sites.

Number of sites, N: 1,000.

Number of extant taxa, M: 512.

Baseline matrix of exchangeabilities, $\mathbb{S}$: WAG (Whelan and Goldman, 2001).

Vector of amino acids' equilibrium frequencies, Π: WAG (Whelan and Goldman, 2001).

Evolution tree topology, τ: Balanced. That is, an evolutionary tree with equal branch lengths where each ancestor in every generation produced exactly two offsprings. An example is presented in Figure 4.1.

Vector that contains the branch lengths, $\mathbf{t}$: A balanced phylogeny with M terminal nodes has $2M - 2$ branches. The default lengths of those branches were set as 0.05. That is, $t_a = 0.05$ for every $t_a \in \mathbf{t}$, where $a = 1, 2, \ldots, 2M - 2$.

Set of foreground branches, $\mathrm{Br}^{(F)}$: Specified as 25% (= 128) of the terminal branches. The selection process involved randomly sampling a terminal branch on the phylogeny and then selecting every second branch afterwards until the required number of foreground branches was obtained.

Site-specific set of target residues, $\mathbf{y}_n$: Set of a single element for each site $n \in \{1, \ldots, N\}$. Each element was a random sample from the space of all possible 20 amino acid residues, $\mathcal{S} = \{\mathtt{A}, \mathtt{C}, \ldots, \mathtt{Y}\}$.

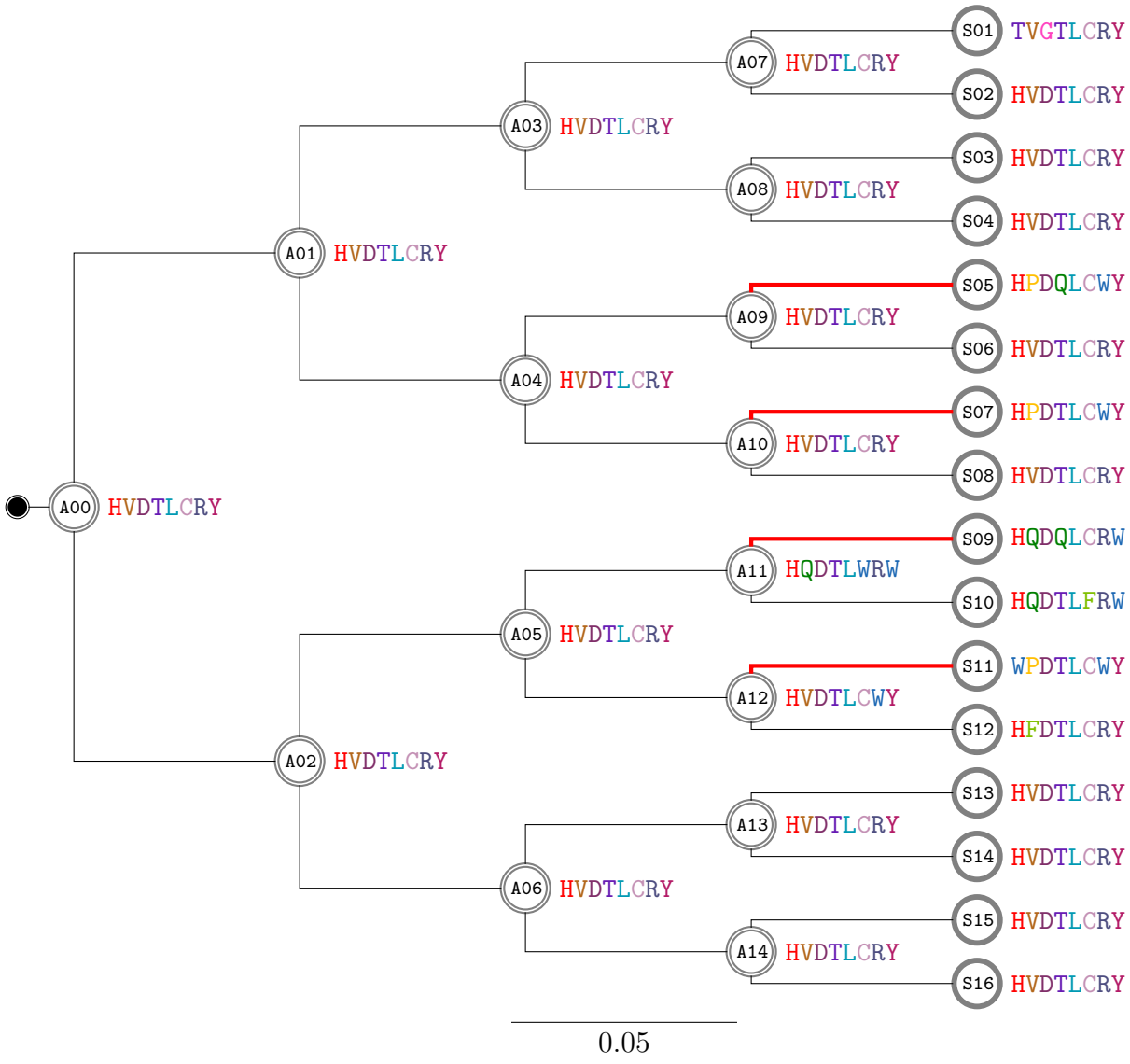


Figure 4.1: A 16-taxa balanced phylogeny with a simulated 8-site sequence at each node. Each branch length was set as 0.05. Foreground branches are indicated in bold. The site-specific sets of the random target amino acid was: $\big(\{\mathtt{W}\}\{\mathtt{P}\}\{\mathtt{D}\}\{\mathtt{Q}\}\{\mathtt{G}\}\{\mathtt{W}\}\{\mathtt{W}\}\{\mathtt{D}\}\big)$.

Site-specific rate multiplier, $\mathcal{A}_n$: Initially, alignments were simulated without rate variation across sites. That is, $\mathcal{A}_n = 1$ for each site $n \in \{1, \ldots, N\}$. The effect of site-to-site rate variation was separately investigated.

Site-specific bias parameter, $\mathcal{B}_n$: The proportion of sites subjected to EDS in each alignment was set to 10%. That is, in an alignment of 1,000 sites, 900 sites were simulated with $\mathcal{B}_n = 0$ while 100 sites were generated with a positive parameter value, $\mathcal{B}_n = 2$ say. The value of the positive bias value was made to vary from one alignment to another with values chosen from the set $\{2, 5, 10, 30, 50, 100, 300, 500, 1000\}$.

4.1.2 Default settings for inferential procedure

At this phase of the study, some of the parameter settings used for the alignment generation are assumed unknown. The primary intention is to infer the bias parameter and the set of target residues for each site. This subsection contains details of the default settings of the parameters needed to achieve this goal.

The number of sites and the number of extant taxa are evident from observation of the alignment. The baseline matrix of exchangeabilities, $\mathbb{S}$, was assumed known for most of the study. A scenario where $\mathbb{S}$ was assumed unknown was investigated in one of the analyses. The equilibrium frequencies of amino acid residues were estimated from the alignment. The tree topology and the foreground branches were assumed known. Maximum likelihood estimates of the branch lengths were obtained from the alignment by optimising the background and foreground models assuming homogeneous site-specific rates. That is, $\mathcal{A}_n$ and $\mathcal{B}_n$ were fixed as one and zero respectively in the $\mathbb{Q}^{nB}$ and the $\mathbb{Q}^{nF}$ models. The next three paragraphs describe how rates, $\mathcal{A}_n$ and $\mathcal{B}_n$, and the target residues, $\mathbf{y}_n$, were handled.

Bias parameter categories, $\mathbb{B}$: Inferences about the true $\mathcal{B}_n$ values were based on pre-specified category values that were generated using the discretisation function from the FUBAR paper. The number of categories, $K_{\mathbb{B}}$, used was set as twenty. That is

$$\begin{aligned}\mathbb{B} &= \left\{\mathbb{B}^{(1)}, \mathbb{B}^{(2)}, \ldots, \mathbb{B}^{(20)}\right\} \\ &= \{0.00,\ 0.07,\ 0.14,\ 0.21,\ 0.29,\ 0.36,\ 0.43,\ 0.50,\ 0.57,\ 0.64, \\ &\quad\ 0.71,\ 0.79,\ 0.86,\ 0.93,\ 1.00,\ 1.39,\ 4.14,\ 11.58,\ 26.09,\ 50.00\}.\end{aligned}$$

Rate multiplier categories, $\mathbb{A}$: The following pre-defined site-specific rate categories were used to account for among-site rate heterogeneity

$$\begin{aligned}\mathbb{A} &= \left\{\mathbb{A}^{(1)}, \mathbb{A}^{(2)}, \ldots, \mathbb{A}^{(20)}\right\} \\ &= \{0.04,\ 0.07,\ 0.14,\ 0.21,\ 0.29,\ 0.36,\ 0.43,\ 0.50,\ 0.57,\ 0.64, \\ &\quad\ 0.71,\ 0.79,\ 0.86,\ 0.93,\ 1.00,\ 1.39,\ 4.14,\ 11.58,\ 26.09,\ 50.00\}.\end{aligned}$$

Site-specific hypothesised set of target residues, $\mathbb{Y} = \left\{\mathbb{Y}^{(n)}\right\}$: To be able to infer the target amino acids it is necessary to test for directional selection towards certain residues. Initially, all the twenty possible amino acids were tested at each site before focusing on only the residues observed at each site.

A total of $K = 400$ $(= K_{\mathbb{A}} \times K_{\mathbb{B}})$ rate categories were used as the default setting. That is, inference about the rate category for the nth site was based on the set of pre-defined categories $\mathbb{Z}$, where

$$\begin{aligned}\mathbb{Z} &= \left\{\left(\mathbb{A}^{(1)}, \mathbb{B}^{(1)}\right), \ldots, \left(\mathbb{A}^{(1)}, \mathbb{B}^{(20)}\right), \left(\mathbb{A}^{(2)}, \mathbb{B}^{(1)}\right), \ldots, \left(\mathbb{A}^{(20)}, \mathbb{B}^{(20)}\right)\right\} \\ &= \{(0.04, 0.00), \ldots, (0.04, 50.00), (0.07, 0.00), \ldots, (50.00, 50.00)\}.\end{aligned}$$

The phylogeny presented in Figure 4.1 has a simulated alignment comprised of 16 observed sequences with 4 sites subjected to episodic directional selection (EDS). The phylogeny is balanced and it has 4 foreground branches shown in bold. The unobserved ancestral sequences are also shown at the internal nodes on the tree. Ancestral sequences were used for calculating the number of visible substitutions toward the target residues along the foreground branches, ν_n, for each site n. Murrell et al. (2012a) demonstrated that the power of the EDEPS model to detect sites under EDS is correlated with ν_n.

An example of how to calculate ν_n for Figure 4.1 is presented in Table 4.1. The site-specific sets of target residues $\mathbf{y}_1$, $\mathbf{y}_2$, $\mathbf{y}_3$, $\mathbf{y}_4$, $\mathbf{y}_5$, $\mathbf{y}_6$, $\mathbf{y}_7$ and $\mathbf{y}_8$ were {W},{P},{D},{Q},{G},{W},{W} and {D} respectively. The pairs of sequences in the table are those of the terminal foreground branches and their most recent ancestors. A substitution is said to be visible at site n along a foreground branch if the amino acid residue at that site is an element of $\mathbf{y}_n$ and different from the residue that occurred at the same site in the sequence of its ancestor.

At the inference phase, the null, $\mathtt{H}_0$, and alternative, $\mathtt{H}_1$, hypotheses considered at each site are

$\mathtt{H}_0$: The rate of evolution toward the target residues does not differ between background and foreground branches. That is, $\mathcal{B}_n = 0$.

Table 4.1: Illustrative example of how to compute the total number of visible substitutions along foreground branches. This computation corresponds to the phylogeny in Figure 4.1.

	Target sequence							
	W	P	D	Q	G	W	W	D
Most recent ancestor: A09	H	V	D	T	L	C	R	Y
Foreground taxa: S05	H	P	D	Q	L	C	W	Y
Visible substitutions	0	1	0	1	0	0	1	0
Most recent ancestor: A10	H	V	D	T	L	C	R	Y
Foreground taxa: S07	H	P	D	T	L	C	W	Y
Visible substitutions	0	1	0	0	0	0	1	0
Most recent ancestor: A11	H	Q	D	T	L	W	R	W
Foreground taxa: S09	H	Q	D	Q	L	C	R	W
Visible substitutions	0	0	0	1	0	0	0	0
Most recent ancestor: A12	H	V	D	T	L	C	W	Y
Foreground taxa: S11	W	P	D	T	L	C	W	Y
Visible substitutions	1	1	0	0	0	0	0	0
Site-specific count of favoured substitution along foreground branches, ν_n	1	3	0	2	0	0	2	0

$\mathtt{H}_1$: The rate of evolution toward the target residues along the foreground branches is greater than that on the background branches. That is, $\mathcal{B}_n > 0$.

The null hypothesis was rejected for sites with Bayes factors greater than or equal to 100 as it was done in Murrell et al. (2013).

4.2 Diagnostic analyses for the CGS algorithm

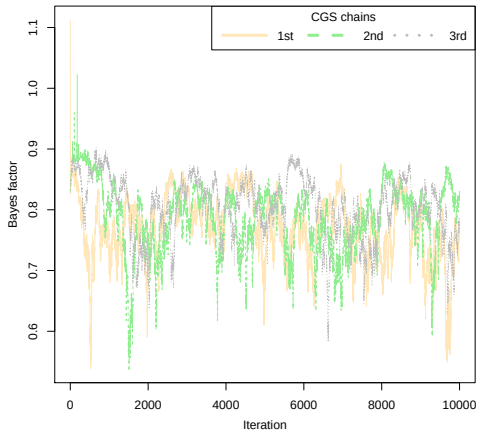
The CGS algorithm is guaranteed to converge to the true posterior distribution if it is allowed to run for long enough. The number of samples that

defines "long enough", referred to as the burn-in size, varies for different analyses and needs to be determined. Also, every Gibbs sample depends on the value of the previous sample. Therefore, to obtain a random sample from the posterior distribution, the chain needs to be sampled at intervals. The length of the sampling interval, referred to as the thinning size, needs to be determined in order to decorrelate the samples. The simulation analyses undertaken to decide on appropriate burn-in and thinning sizes are discussed in this section.

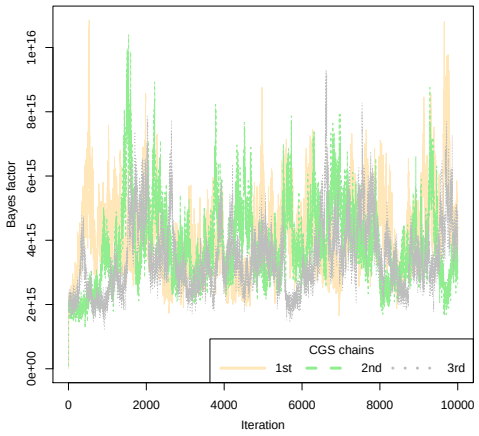
Due to the expensive computational cost of implementing the CGS algorithm, some of the default specifications were simplified for the analyses presented in this section. An alignment of $N = 1,000$ sites was simulated, where 100 sites were subjected to EDS. Of the 100 sites under positive EDS pressure, 11 were simulated with $\mathcal{B}_n = 2$, another 11 sites had $\mathcal{B}_n = 5$, and so on for values of $\mathcal{B}_n$ including 10, 30, 50, 100, 300 and 500. The remaining 12 sites under positive EDS were simulated with $\mathcal{B}_n = 1,000$.

Still on data generation, the site-specific sets of target residues were fixed as $\{\mathtt{A}\}$. That is, $\mathbf{y}_1 = \mathbf{y}_2 = \ldots = \mathbf{y}_{1000} = \{\mathtt{A}\}$. An equiprobable baseline model was used with exchangeabilities $s_{ij} = 1$ and equilibrium frequencies $\pi^{(i)} = 0.05$, for $i, j \in \{\mathtt{A}, \mathtt{C}, \ldots, \mathtt{Y}\}$. For data modelling, $K_{\mathbb{A}}$ was set as 1 while $K_{\mathbb{B}} = 20$ such that $\mathbb{Z} = (\{1, 0.00\}, \{1, 0.07\}, \ldots, \{1, 50.00\})$.

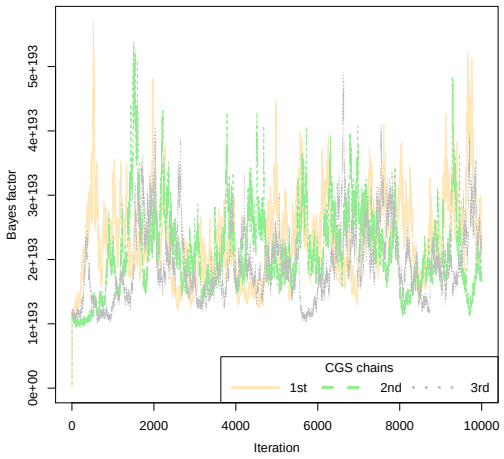
Three randomly and independently initialised Markov chains were generated for the diagnostics presented. Each chain contained 10,000 draws. Trace plots of the chains of the Bayes factors for three randomly chosen sites are presented in Figure 4.2. The presented sites are governed by different $\mathcal{B}_n$ values. It appears from the figure that convergence to the stationary posterior distribution was generally achieved prior to the $2,000$th iteration. The results for the other sites were similar to these.



(a) $\mathcal{B}_n = 0$



(b) $\mathcal{B}_n = 50$

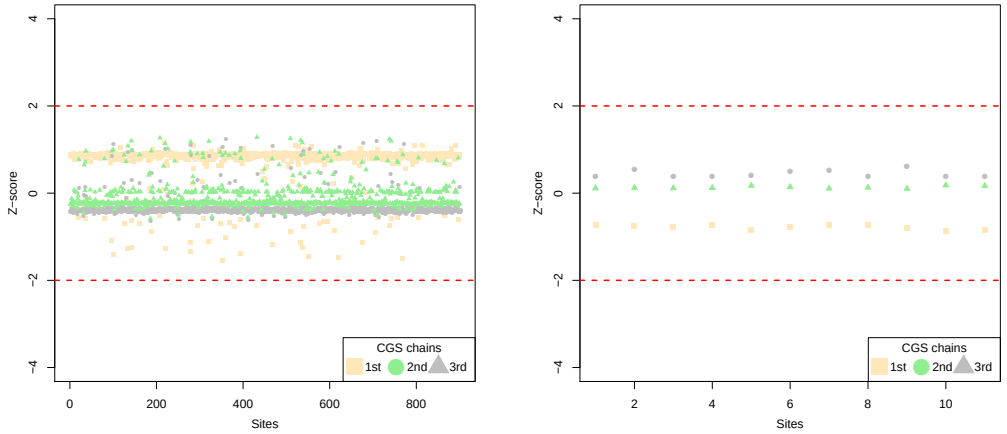


(c) $\mathcal{B}_n = 1000$

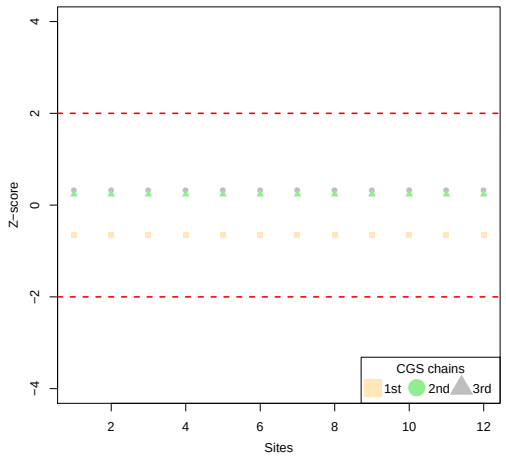
Figure 4.2: Trace plots showing the sampled Bayes factors for three separate CGS chains over $10,000$ iterations for three randomly chosen sites, each with separate $\mathcal{B}_n$ values.

The plot in Figure 4.3 is the Geweke (Geweke, 1992) diagnostic plot of the Bayes factors for all sites and $\mathcal{B}_n \in \{0, 50, 1000\}$ for the three chains. The Bayes factors for sites with $\mathcal{B}_n = 1,000$ were often around 10^{200} and computation of Geweke statistics requires sums of squares of these estimates, which are computationally undefined. As a result, the log Bayes factors for $\mathcal{B}_n = 1,000$ were plotted instead.

The plots, and the remaining diagnostic outputs that are referred to in this



(a) $\mathcal{B}_n = 0$ (b) $\mathcal{B}_n = 50$



(c) $\mathcal{B}_n = 1000$

Figure 4.3: Geweke plots for three separate CGS chains corresponding to different $\mathcal{B}_n$ values.

research, were generated with the `coda` (Plummer et al., 2006) package in `R`. The Geweke statistics were intended to support the visual observation from Figure 4.2. The Geweke test statistic has a standard Normal distribution and it is governed by the null hypothesis that the sample for a site is drawn from a stationary distribution, after correcting for autocorrelation. Test statistics that lie in the range $(-2, +2)$ indicate that there is insufficient evidence against the null hypothesis.

All of the chains shown in Figure 4.3 were generated with burn-in sizes of not more than $2,000$. The corresponding plots for other $\mathcal{B}_n$ values were

observed to behave similarly. Therefore, since all of the test statistics lie comfortably within the ± 2 cut-off lines, there is no sufficient evidence to reject the hypothesis that the chains had attained stationarity at the time of the $2,000$th iteration. Therefore, a burn-in size of 20% (i.e. 2000/10000) of the initial Gibbs samples drawn is sufficient.

The plots of the partial autocorrelation factors (PACFs) shown in Figure 4.4 are for the same sites that were used for Figure 4.2. The data used for the plots excludes the burn-in sample. The purpose of the plots is to aid the decision on an optimal thinning size that will minimise the autocorrelation among elements of each of the sampled chains. The plots suggest that a thinning size of 20 is appropriate. Similar deductions were evident from the other sites that are not shown. Therefore, 20 will be adopted as the thinning size for the main analyses.

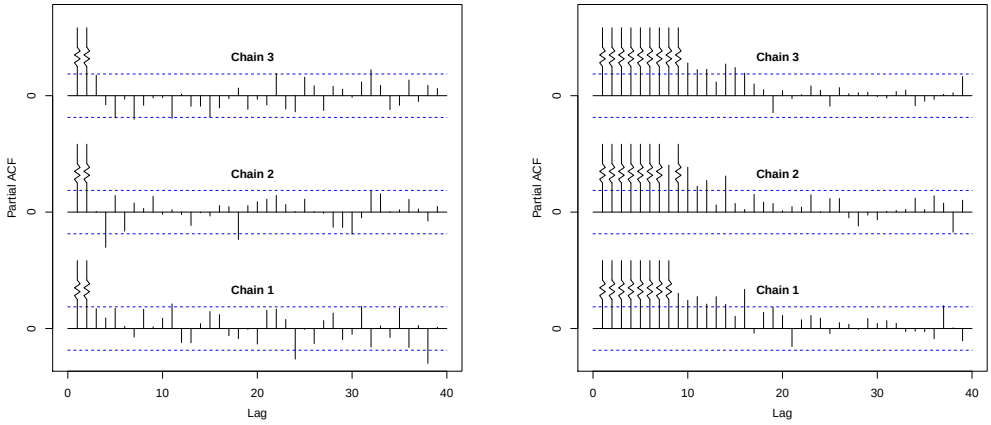
Based on the conclusions from this section, independent samples from CGS implementations were generated by first discarding 20% of the elements of the full chain, and then sampling at intervals of size 20. Therefore, for each of the 1,000 independent CGS samples that were used in this research, the length of the corresponding chains were 25,000 (i.e. $1,000 \times 20/(1 - 0.20)$).

4.3 Comparative analyses

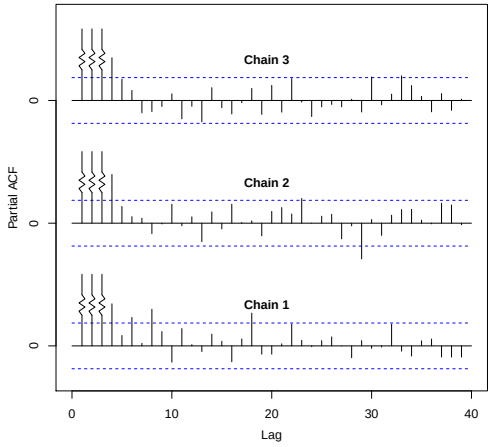
Given a sequence alignment with sites that are suspected to have experienced episodic directional selection, the algorithms considered in this research are designed to detect the sites that are under selection pressure and the set of residues that are favoured at each site. The performances of the CVB, the CVB0 and the CGS algorithms are compared in this section. Since the CGS does converge to the true posterior distribution while the variational Bayes algorithms only provide approximations to the true distribution, the CGS algorithm serves as a benchmark for the CVB and the CVB0 algorithms.

4.3.1 Simple illustrative example

An example of how the main statistics that are reported in this section are computed is presented in this subsection. The analysed alignment is the same data presented in Figure 4.1, which is replicated in Table 4.2. The analysis proceeds as for a single simulation run, with site multiplier $\mathcal{A}_n = 1$ and bias parameter $\mathcal{B}_n = 50$ for $n = 1, 2, \ldots, 8$. The site-specific sets of target amino



(a) $\mathcal{B}_n = 0$ (b) $\mathcal{B}_n = 50$



(c) $\mathcal{B}_n = 1000$

Figure 4.4: Plots of partial autocorrelation factors corresponding to the Bayes factor estimates from three separate CGS chains after discarding the first 2,000 samples. The shown plots are for randomly selected sites characterised by separate $\mathcal{B}_n$ values.

acids, $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_8$ were {W}, {P}, {D}, {Q}, {G} and {W} respectively. Due to the small number of sites, the data generation was such that 50% of the 8 sites had $\mathcal{B}_n = 0$. Table 4.2 indicates the $\mathcal{B}_n$ that governs each site.

Table 4.2: Simulated protein sequence alignment analysed in subsection 4.3.1 with the bias parameter, $\mathcal{B}_n$, and set of target residues, $\mathbf{y}_n$, presented for each site.

		Site							
		1	2	3	4	5	6	7	8
	$\mathcal{B}_n$	50	50	0	50	0	0	50	0
	$\mathbf{y}_n$	{W}	{P}	{D}	{Q}	{G}	{W}	{W}	{D}
Extant taxa	S01	T	V	G	T	L	C	R	Y
	S02	H	V	D	T	L	C	R	Y
	S03	H	V	D	T	L	C	R	Y
	S04	H	V	D	T	L	C	R	Y
	S05	H	P	D	Q	L	C	W	Y
	S06	H	V	D	T	L	C	R	Y
	S07	H	P	D	T	L	C	W	Y
	S08	H	V	D	T	L	C	R	Y
	S09	H	Q	D	Q	L	C	R	W
	S10	H	Q	D	T	L	F	R	W
	S11	W	P	D	T	L	C	W	Y
	S12	H	F	D	T	L	C	R	Y
	S13	H	V	D	T	L	C	R	Y
	S14	H	V	D	T	L	C	R	Y
	S15	H	V	D	T	L	C	R	Y
	S16	H	V	D	T	L	C	R	Y

Computation of all the reported statistics in the main analyses depends on Bayes factors. Therefore, Table 4.3 contains the Bayes factors corresponding to amino acids tested for evidence of EDS, $\mathbb{Y}_n$, at each site of the alignment in Table 4.2.

The bold values in Table 4.3 are the Bayes factors for the true target residues. For example, the bold 2.505 in row W at the first site indicates that the target residue at that site $\mathbf{y}_1 = \{\mathtt{W}\}$. Not all the amino acids tested for evidence of EDS were observed at each site. The cells that correspond to the tested target residues that were actually observed are shaded. For example, the fifth site only contained residue L. The underlined Bayes factors are those that imply positive episodic directional selection i.e. the Bayes factors that are at least 100. The following paragraphs are dedicated to explaining how the summary statistics presented in the rest of this chapter are computed.

Table 4.3: Bayes factors for each of the 20 elements of the default hypothesised set of target residues at every site of the alignment in Table 4.2. The third row contains the bias parameter used during generation of the alignment. Bold values correspond to the true targets, shaded cells indicate observed residues and underlined values imply positive selection.

		Site							
		1	2	3	4	5	6	7	8
	$\mathcal{B}_n$	50	50	0	50	0	0	50	0
$\mathbb{Y}_n$	A	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	C	0.940	0.769	0.972	0.939	1.032	2.146	0.908	0.973
	D	0.953	0.779	**1.788**	0.951	1.045	0.985	0.920	0.986
	E	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	F	1.019	0.839	1.036	1.019	1.060	1.027	0.999	1.036
	G	1.008	0.915	1.018	1.007	**1.032**	1.018	0.993	**1.018**
	H	0.935	0.841	1.052	1.017	1.116	1.052	0.984	1.053
	I	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	K	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	L	0.974	0.792	1.009	0.973	1.518	1.009	0.940	1.010
	M	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	N	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	P	0.858	**3809.028**	0.875	0.858	0.902	0.875	0.840	0.876
	Q	0.844	0.950	0.864	**68.731**	0.894	0.864	0.824	0.864
	R	1.113	0.927	1.153	1.111	1.225	1.153	0.572	1.153
	S	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	T	1.062	0.874	1.100	0.730	1.169	1.100	1.026	1.100
	V	1.051	0.670	1.085	1.050	1.144	1.085	1.017	1.085
	W	**2.505**	0.673	0.798	0.777	0.832	**0.798**	**2638.693**	0.829
	Y	0.962	0.792	0.995	0.961	1.054	0.995	0.930	1.569

Power: This is defined as the probability of detecting a target residue given that it is truly under episodic directional selection (EDS) pressure. It is the number of cases in the intersection of the positive predictions and the true bias parameter scenario, divided by the number of the true cases for value of $\mathcal{B}_n = 50$. In Table 4.3, this corresponds to

$$\frac{\text{Number of bold and underlined Bayes factors under columns with } \mathcal{B}_n = 50}{\text{Number of columns with } \mathcal{B}_n = 50} = \frac{2}{4}.$$

Proportion of false positives: Proportion of false positives is often referred to as the probability of `Type 1` error. It is the probability of incorrectly inferring EDS when the actual value of $\mathcal{B}_n$ is zero. It was computed as the number of target amino acids governed by $\mathcal{B}_n = 0$ that were incorrectly described with a positive bias parameter divided by the total number of residues governed by $\mathcal{B}_n = 0$. Ideal inferences produce zero false positive ratios. With respect to Table 4.3, this statistic can be calculated as

$$\frac{\text{Number of bold and underlined Bayes factors under columns with } \mathcal{B}_n = 0}{\text{Number of columns with } \mathcal{B}_n = 0} = \frac{0}{4}.$$

Positive predictive value, PPV: The PPV refers to the proportion of the predictions of positive EDS that are correct. It measures the reliability of positive predictions. This ratio is computed as the number of positively selected residues that are correctly predicted, divided by the total number of positive predictions. In an ideal case, PPV should be equal to one. The PPV estimate from the results in Table 4.3 is

$$\frac{\text{Number of bold and underlined Bayes factors under columns with } \mathcal{B}_n = 50}{\text{Number of underlined Bayes factors}} = \frac{2}{2}.$$

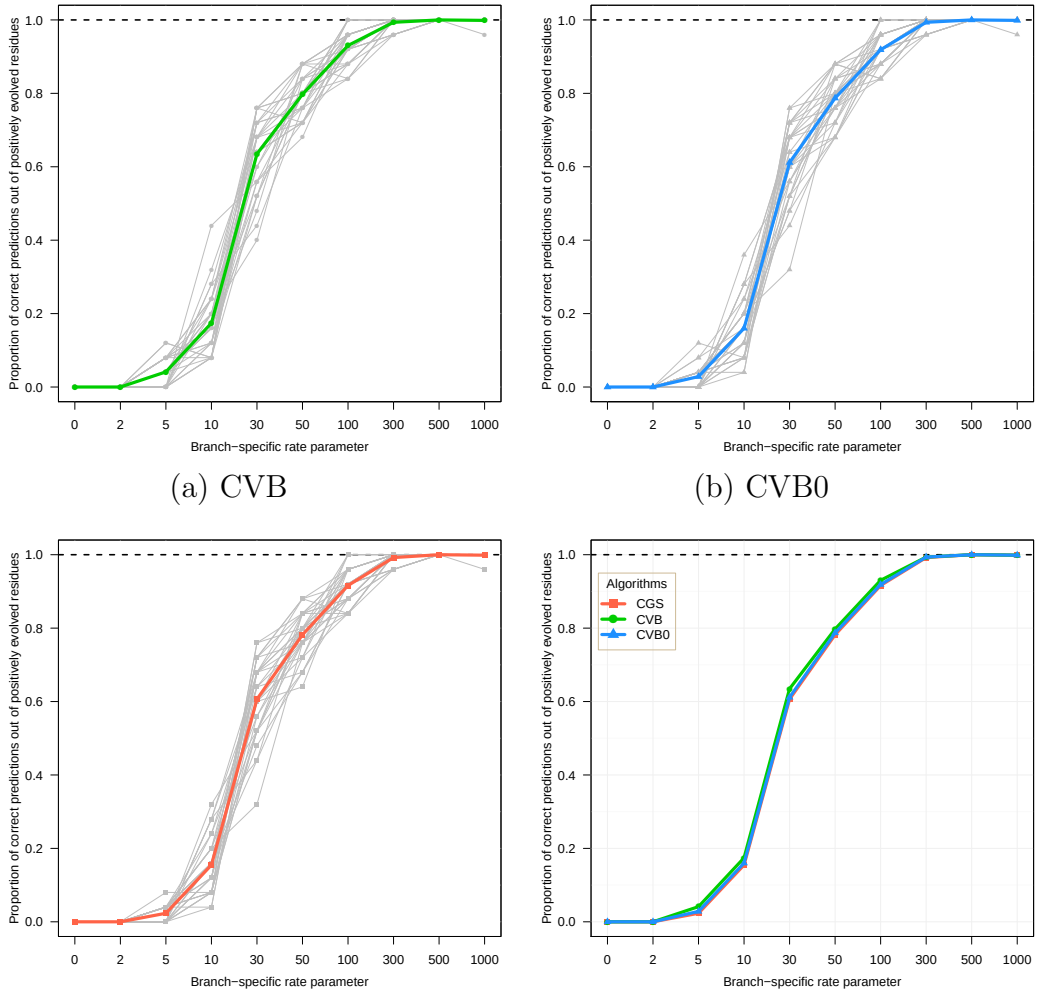
The Bayes factors in the unshaded cells in Table 4.3 each correspond to a situation where there is no information for or against EDS. As a result, the corresponding Bayes factors are expected to be one. The computation of this statistic (as well as power, false positives and PPV) in terms of residues, instead of sites, is to account for cases where multiple residues are under episodic directional selection at any particular site.

4.3.2 Main analyses

The results summarised in Table 4.4 and Figure 4.5 were obtained from analysing 279 alignments that each had 250 sites. The alignments had 10% of their sites subjected to EDS where the value of the associated positive bias parameter was varied among alignments. The positive bias parameter was chosen from the set $\{2, 5, 10, 30, 50, 100, 300, 500, 1000\}$ and it was ensured that each value described equal number of alignments. No rate variation was allowed among sites. For data modelling, the number of the pre-defined site multiplier categories was one while the number of the pre-specified bias parameter categories was twenty.

Table 4.4: Five-number summary and mean of the false positive ratio in the absence of site-specific rate multiplier effect. The data used for the different algorithms is the same as for Figure 4.5.

		Min.	Quantiles			Max.	Mean
			25%	50%	75%		
Algorithms	CVB	0.000	0.000	0.000	0.000	0.004	0.000
	CVB0	0.000	0.000	0.000	0.000	0.000	0.000
	CGS	0.000	0.000	0.000	0.000	0.000	0.000



(a) CVB (b) CVB0

(c) CGS (d) All algorithms

Figure 4.5: Power curves over various bias parameters, $\mathcal{B}_n$, for the three algorithms analysed.

Table 4.4 shows the five-number summary and mean of the proportion of false positives for the CGS, the CVB and the CVB0 algorithms. The maximum false positive proportions in the table show that the CVB0 is more similar to the CGS approach. Only two out of the 279 false positive proportions for the CVB were actually 0.004, other proportions were zero. Therefore, all the approaches had substantially low false positive proportions, in the absence of site-specific rate multiplier effect.

The power curves in Figure 4.5 were obtained from the same analysis that generated Table 4.4. The distinctly coloured curve on each of the plots is the average of the corresponding grey curves. Plot 4.5(d) presents all of the averaged curves and it shows no substantial difference among the algorithms.

The plots in Figure 4.6 show the site-specific number of visible substitutions toward the target residue along the foreground branches, ν_n, against the associated log Bayes factors. The dots represents the actual values while the lines are the corresponding regression lines. Both plots are the same except that the right-hand plot shows only the regression lines. The figure supports the claim by Murrell et al. (2012a), that the power to detect sites under EDS strongly depends on the amount of information, in terms of ν_n, present in the alignment. The summarised results were obtained from the CGS implementation but the results from the other algorithms were similar.

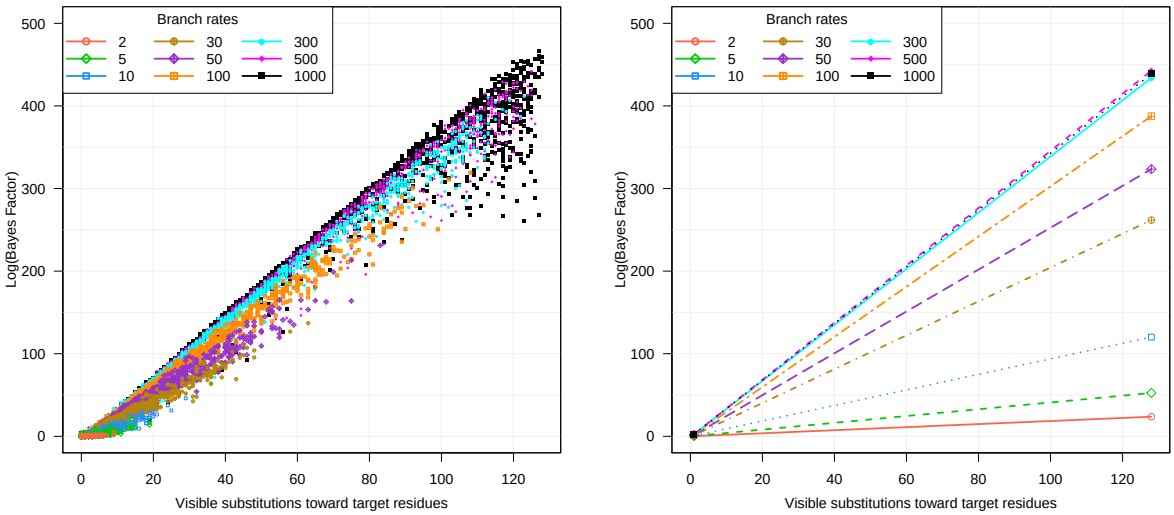


Figure 4.6: Plots of the number of visible substitutions toward the target residues along the foreground branches against the corresponding log Bayes factors. Both plots are the same except that the right-hand plot excludes the actual values and shows only the regression lines.

Since the ability to infer episodic directional selection depends on the number of substitutions toward the target residue, considering a target residue that is not present at the observed site is not expected to be useful. Figure 4.7 shows the distributions of the site-specific Bayes factors obtained when the target residue was unobserved at a site.

The densities in Figure 4.7 are centred around one, as expected. The Bayes factors from the CVB algorithm appear to be downwardly biased, relative to the benchmark CGS algorithm. The five-number summaries presented in the figure show that the CVB produced relatively heavier tails. The Bayes factors from the CVB0 algorithm, on the other hand, appears to match the CGS very closely.

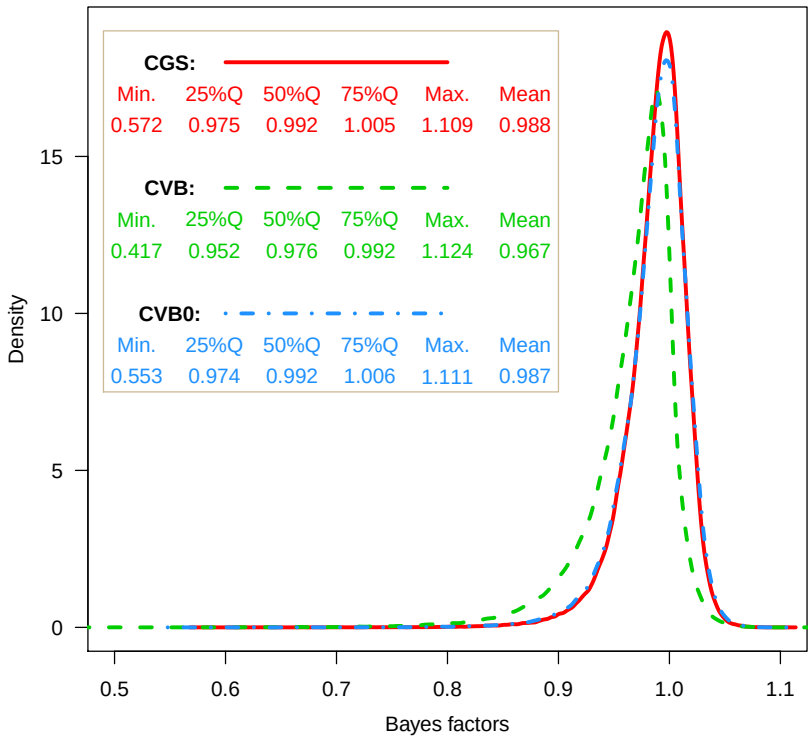


Figure 4.7: Distributions of the Bayes factors for target residues that were not observed.

The box plots presented in Figure 4.8 summarise the proportion of sites that are truly under EDS out of those inferred to be evolving as such, computed for each independent simulation run. Results from the different algorithms are presented. With outlying minima at approximately 95%, the figure shows that, when a target residue is identified by any of the algorithms, the inference is highly reliable.

The plots in Figure 4.9 compare the computation times (in seconds) required by each of the algorithms. The plots show that CGS is the most computationally expensive algorithm as expected, while CVB0 is computationally the cheapest. So, even though the results from the three algorithms were so similar, the CVB0 algorithm is recommended.

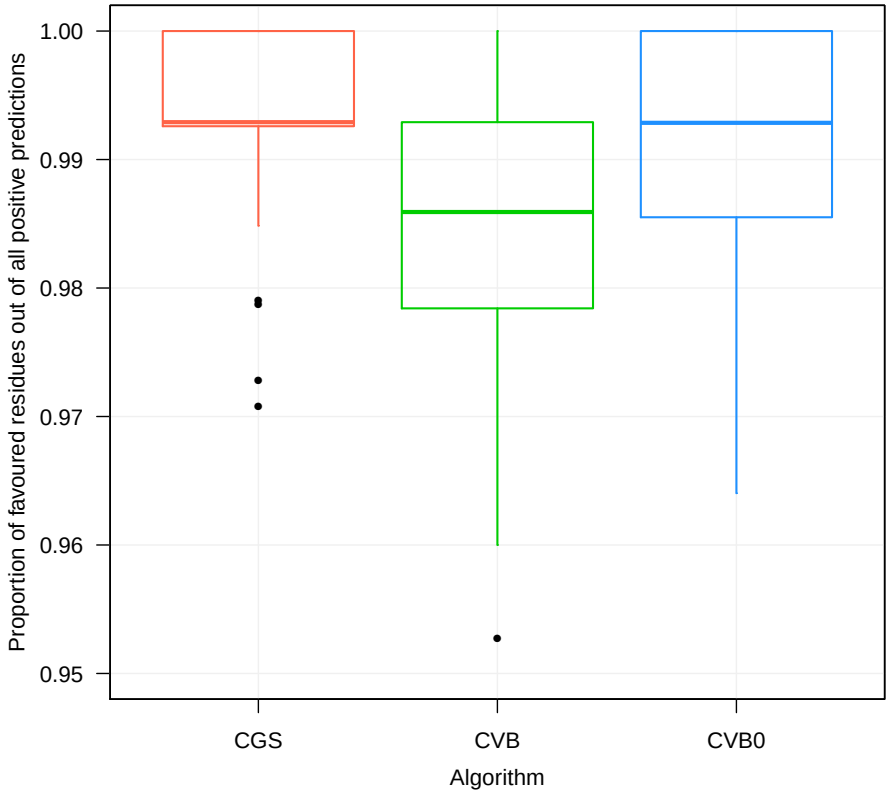


Figure 4.8: Distributions of PPV for models without site-specific rate multiplier effect.

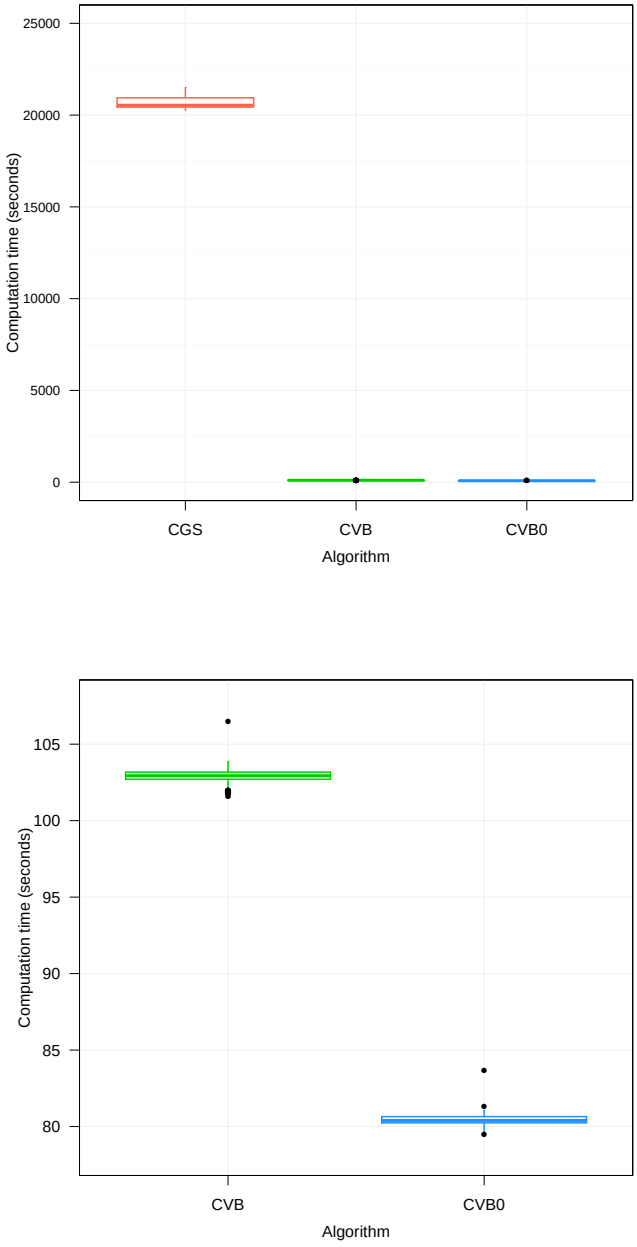
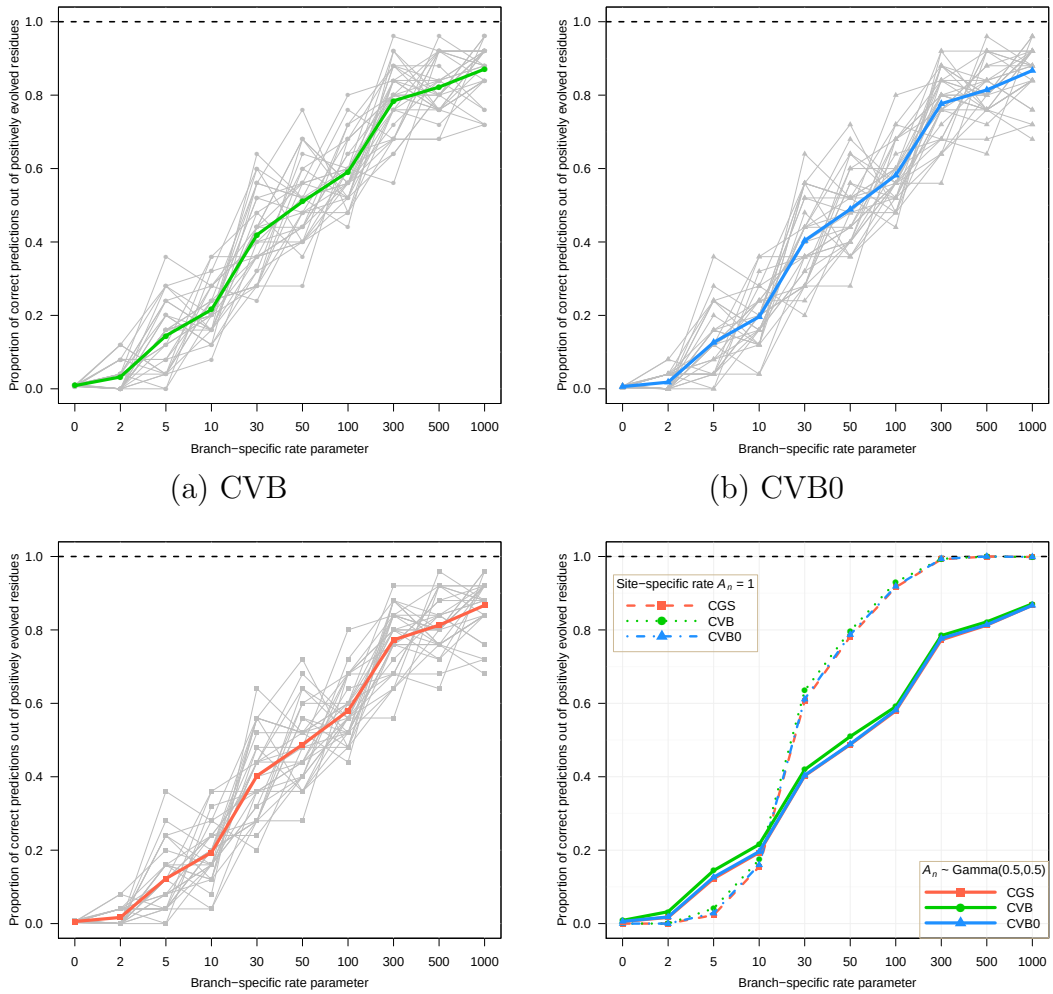


Figure 4.9: Comparison of the computation time in seconds of the algorithms implemented. The presented plots are similar, except that the plot at the bottom excludes the CGS approach.

Incorporation of site-specific rate multiplier. Figure 4.10 shows power curves similar to those in Figure 4.5 except that the site-specific rate multiplier, $\mathcal{A}_n$, was independently and randomly sampled from a `Gamma`$(0.5, 0.5)$ distribution for each site when generating the alignment. Similar to Figure 4.5, the illustrations are based on 31 complete simulation sets. The results from these analyses are expected to lead to less accurate inference, relative to when $\mathcal{A}_n = 1$, because rate multiplier effect was not accommodated in the model.



(a) CVB (b) CVB0

(c) CGS (d) All algorithms

Figure 4.10: Power curves obtained from alignments generated with heterogeneous site-specific rate multiplier. The thick curves in Plots (a), (b) and (c) are averages of the other curves in the plots. Plot (d) compares the average curves to those from alignments with homogeneous multiplier effect (shown as dashed curves).

As expected, Figure 4.10 shows that inferences obtained from alignments generated using models with site-specific rate multiplier effect resulted in less power and increased variance relative to the results obtained without rate multiplier effect. The summaries in Table 4.5 and Figure 4.11 showing slightly worsened false positive rates also agree with the expectation.

The analyses presented in the next section explore a way to improve on inference in the presence of site-specific rate multiplier effect by incorporating it into the modelling process. The effects of varying some other components

Table 4.5: Summary of false positives for alignments generated with $\mathcal{A}_n \sim \texttt{Gamma}(0.5, 0.5)$.

		Min.	Quantiles			Max.	Mean
			25%	50%	75%		
Algorithms	CVB	0.000	0.004	0.009	0.013	0.036	0.009
	CVB0	0.000	0.000	0.004	0.009	0.031	0.006
	CGS	0.000	0.000	0.004	0.009	0.031	0.006

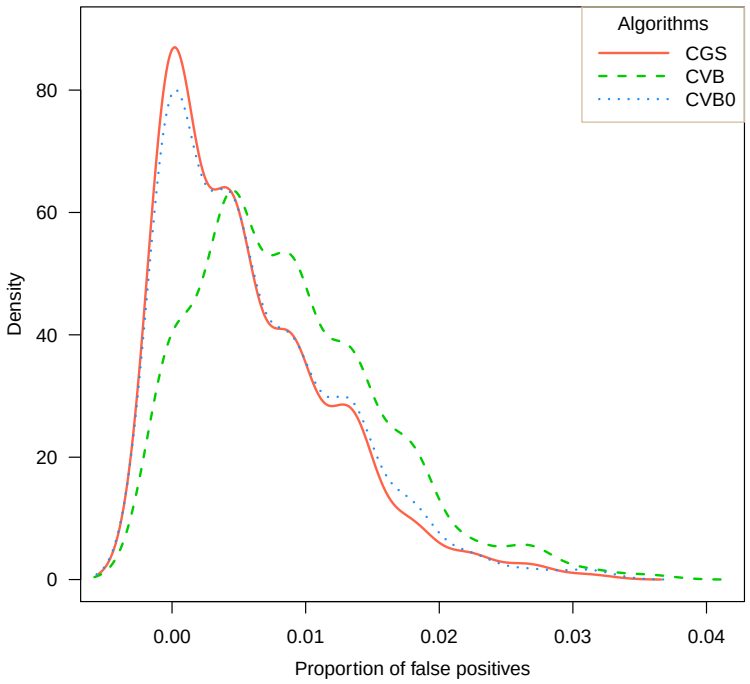


Figure 4.11: False positive rates for alignments generated with $\mathcal{A}_n \sim \texttt{Gamma}(0.5, 0.5)$.

of the default specifications are also examined.

Despite using a cruder zero-order approximation for CVB0, its results surprisingly approximated the CGS results better than the CVB where a more accurate second-order approximation was applied. This point was noted, in the context of topic modelling, by Asuncion (2010) and it was investigated by Sato and Nakagawa (2012) who termed it as the "zero forcing effect". The CVB0 algorithm was also shown to require the least computation time of the three algorithms. Consequently, the CVB0 algorithm was adopted for subsequent analyses.

4.4 Sensitivity analyses

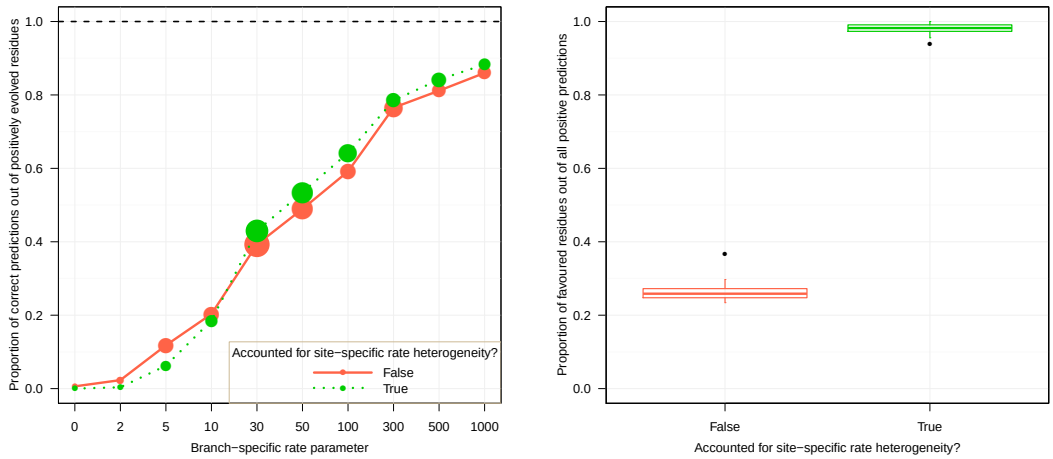
All the analyses that are undertaken in this section employ only the CVB0 algorithm. The goal is to investigate how sensitive inference is to changes in modelling choices. Each subsection is committed to addressing the effect of variation in a component of the model. The results are presented in terms of power curves averaged over multiple independent simulations. The area of the dots on the curves are drawn in proportion to the variance across simulation runs. Box plots of the positive predictive values (PPVs) are also presented for each analysis.

All of the analyses were designed to produce results from 100 independent alignments for each of the various bias values investigated. Due to limited computational resources, in most cases, the number of completed simulations were less than 100. The number of simulation runs that produced the presented results are always noted.

4.4.1 Site-specific rate multiplier

Site-specific rate variation was shown in subsection 4.3.2 to yield lower power. The purpose of this subsection is to investigate the effect of accounting for site-specific rate multiplier during data modelling. Models that account for rate multiplier are expected to perform at least as good as models that do not, since they are less restrictive.

The power curves and PPV box plots in Figure 4.12 were obtained from 38 complete simulations. Construction of alignments was based on the default settings except for the number of sites, which was set as 250 and the site-specific rate multiplier that was independently sampled for each site from



(a) Power curves (b) Positive predictive values

Figure 4.12: Analysis of the effect of accounting for site rate multiplier on episodic directional selection inference, when it is actually present in the alignment. The area of the dots on the curves are proportional to the variance across simulations.

a `Gamma`$(0.5, 0.5)$ distribution. The two curves differ on how the alignments were modelled. The results that generated one of the curves incorporated site rate multiplier using the 20 pre-defined categories set by default while the other curve did not.

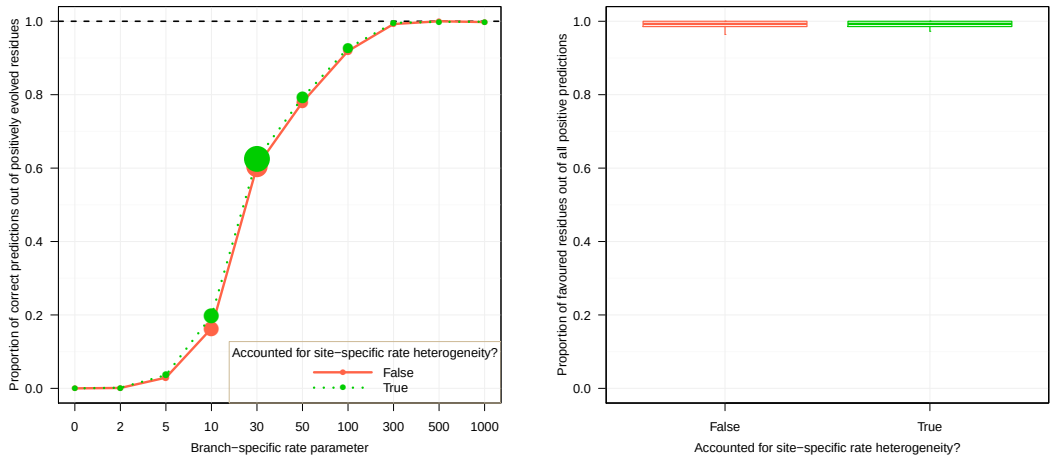
Figure 4.12 shows that incorporating site rate multiplier in data modelling is associated with less variable and improved power as well as substantially better PPVs. The summaries of the associated false positives are presented in Table 4.6.

Table 4.6: False positives obtained from the simulation study that investigated the effect of incorporating site-specific rate multiplier effect when it actually characterises the alignment. The `False` row correspond to results for the modelling process that ignored rate multiplier effect while the `True` row accounted for heterogeneous multiplier effect.

Accounted for rate multiplier	Min.	Quantiles			Max.	Mean	Sd.
		25%	50%	75%			
`False`	0.0025	0.0049	0.0064	0.0077	0.0109	0.0064	0.0019
`True`	0.0000	0.0000	0.0000	0.0000	0.0010	0.0001	0.0002

Table 4.6 shows that accounting for heterogeneous site rate multiplier produced less variable and fewer incorrect predictions of unequal bias values. Modelling alignments characterised by site-specific rate heterogeneity with models that accommodate such heterogeneity therefore leads to improved inference as expected.

Figure 4.13 is an analogue of Figure 4.12 for alignments simulated with equal site rates. The purpose here was to determine the effect of accounting for rate multiplier effect when it is not present in the data. The figure shows strong similarity between the power curves. Table 4.7 presents quantitative summaries of the corresponding false positives. The similarity between the results in the table supports the illustrations in the figure. It is thus claimed that accounting for site-specific rate multiplier effect when it is not present does not adversely affect inferences.



(a) Power curves (b) Positive predictive values

Figure 4.13: Analysis of the effect of accounting for site-specific rate multiplier on episodic directional evolution inference, when such factor does not describe the alignment. The area of the dots on the curves are proportional to the variance across simulations.

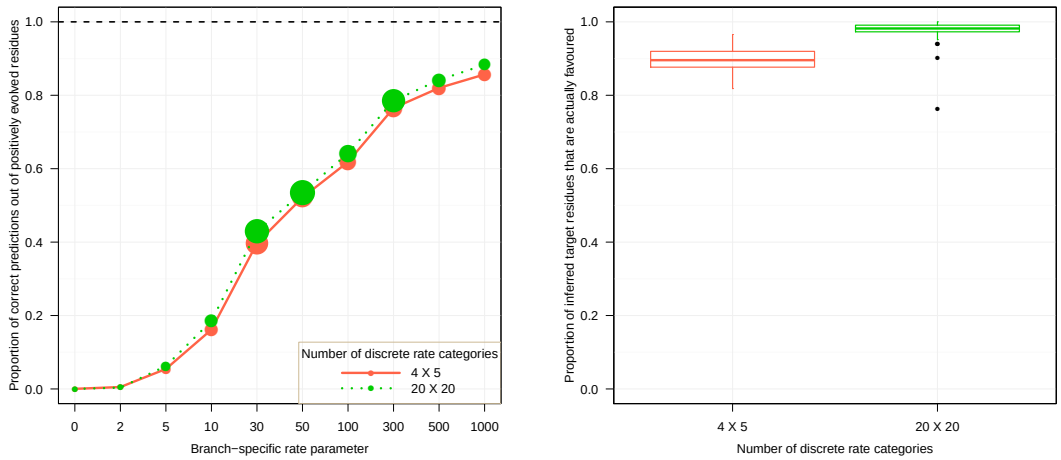
Table 4.7: Quantitative summaries of the false positives obtained from the investigation of how accommodating variation in site-specific rate multiplier in the data modelling phase affects EDS inference. The analysed alignments were characterised by homogeneous site rates. The results in the `False` row correspond to the analysis that ignored the multiplier effect while the `True` row did otherwise.

Accounted for multiplier effect	Min.	Quantiles			Max.	Mean	Sd.
		25%	50%	75%			
`False`	0.0000	0.0000	0.0000	0.0000	0.0005	0.0000	0.0001
`True`	0.0000	0.0000	0.0000	0.0000	0.0005	0.0000	0.0001

4.4.2 Number of rate categories

The default specification uses 20 values for each rate parameter. In this subsection, the effect of using less discrete rate categories is investigated. Two types of analyses were considered and the alignment generation process followed the same process described in Subsection 4.4.1. The results obtained are summarised in Figure 4.14 and Table 4.8.

In one of the analyses that produced the results illustrated in Figure 4.14,



(a) Power curves (b) Positive predictive values

Figure 4.14: Analysis of the effect of the number of rate categories on EDS inference. The area of the dots on the curves are proportional to the variance across simulations.

Table 4.8: Performance comparison between analyses that differ only by the number of rate categories used for alignment modelling. The contents of the table are the five number summary, mean and standard deviation of the rate of false positive predictions.

Number of rate categories	Min.	Quantiles			Max.	Mean	Sd.
		25%	50%	75%			
`4 × 5`	0.0000	0.0000	0.0000	0.0005	0.0015	0.0003	0.0004
`20 × 20`	0.0000	0.0000	0.0000	0.0000	0.0010	0.0001	0.0002

the default number (i.e. 20) of rate categories and the associated pre-defined values were used for data modelling. The other set of results had less number of categories with $\mathbb{A} = \{0.25, 0.50, 1.00, 50.00\}$ and $\mathbb{B} = \{0.00, 0.33, 0.67, 1.00, 50.00\}$. The results presented were obtained from 100 complete simulation runs.

More rate categories is expected to capture the continuity of the `Gamma` distribution that generated the site rates. Thus, the `20 × 20` rate categories are expected to produce better results. Figure 4.14 presents visual summaries of the results obtained from the simulation analyses designed to verify this expectation, while Table 4.8 presents a quantitative summary of the false positive rates. As expected, the results imply that the `20 × 20` analysis resulted in better inferences, in most cases. Similar results were obtained by Murrell et al. (2013).

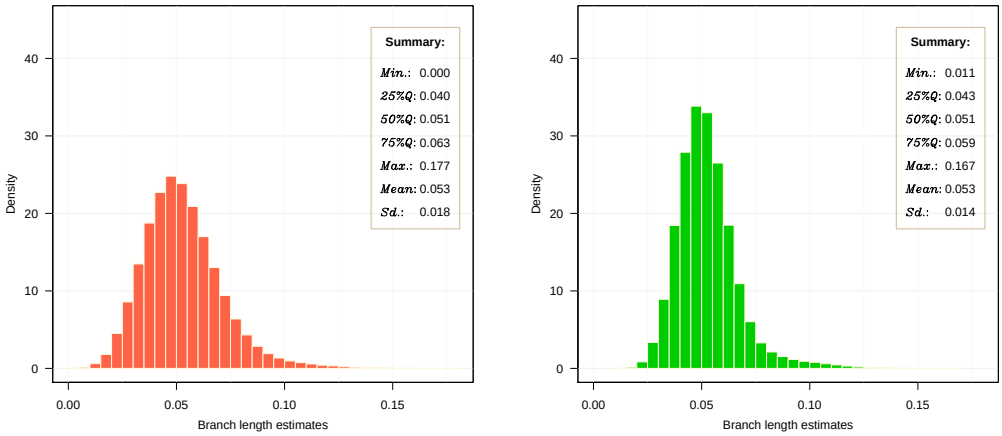
Subsequent analyses in this section use the `20 × 20` rate modelling approach and the alignment generation procedures involve equal site rates. The small and similar values in Tables 4.7 and 4.8 provide good representations of the summaries of false positives obtained from these analyses. Consequently, explicit false positive summaries are no longer always shown. It is claimed that when the bias values do not differ between the foreground and the background branches, the CVB0 algorithm rarely infers otherwise, regardless of the properties of the alignments.

4.4.3 Number of alignment sites

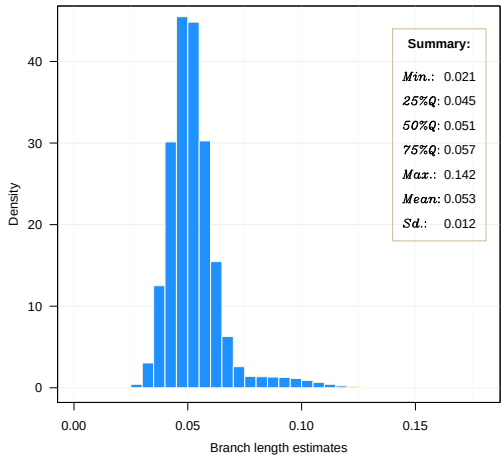
The analyses presented in this subsection are based on alignment with various number of sites. The interest here is to examine how the number of alignment sites affects results. Except for the number of sites, all settings

used were as stated in the default for the data generation and the modelling stages. The analyses were such that the sites that made up shorter alignments were subsets of the longer ones. The proportion of sites exposed to positive episodic directional evolution was maintained at 10% irrespective of the total number of sites in the alignment.

More alignment sites are associated with more accurate branch length estimates. This is illustrated in Figure 4.15 where the distribution and the quantitative summaries of the length estimates are presented.



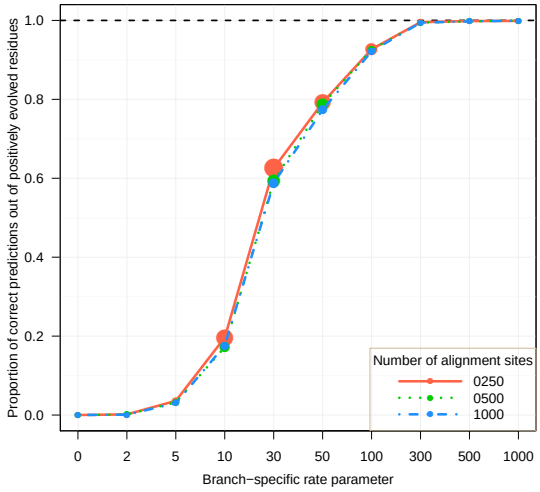
(a) $N = 250$ (b) $N = 500$



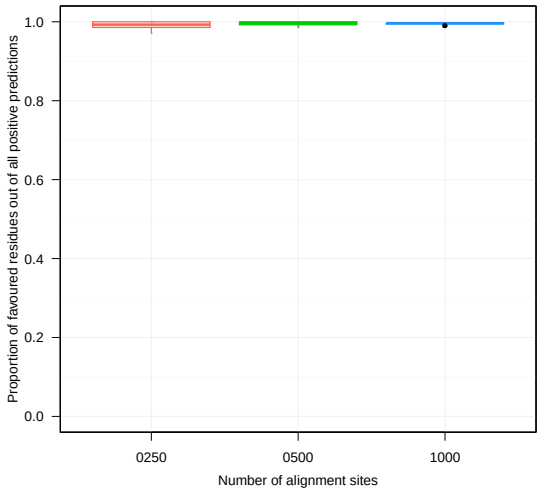
(c) $N = 1000$

Figure 4.15: Visual and quantitative summaries of the distribution of branch length estimates obtained from alignments that differ only by their number of sites, N. The true lengths were all 0.05.

The results presented in Figure 4.15 were from 33 completed simulation runs. The figure shows that, the more the sites in an alignment, the less variable and the closer the estimates of the branch lengths are to their true values (= 0.05). More sites are therefore expected to facilitate more reliable inference about the bias parameters. The results from the undertaken study are shown in Figure 4.16.



(a) Power curves



(b) Positive predictive values

Figure 4.16: Illustration of how inferences are affected by the number of sites in an alignment. The area of the dots on the curves are proportional to the variance across simulations.

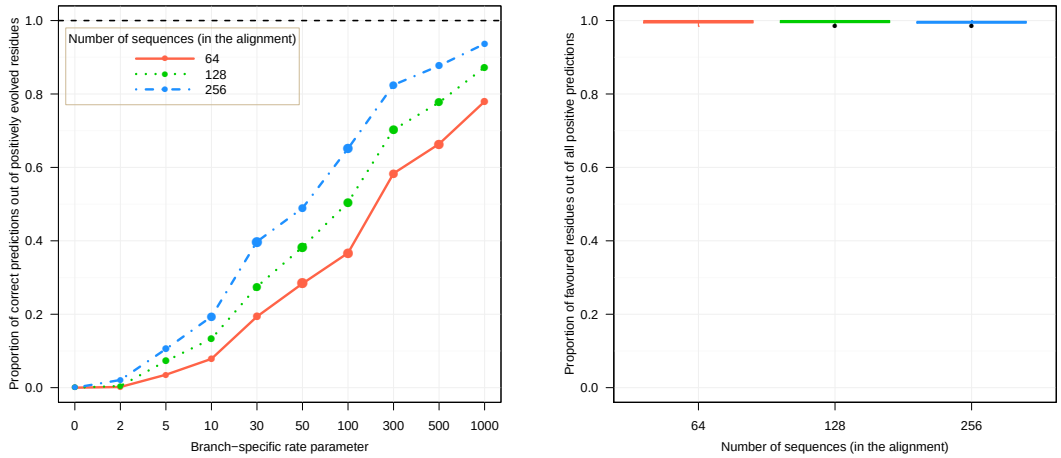
Figure 4.16 presents summary plots of the power and predictive values obtained from the different variants of alignments analysed. The plots show no substantial difference among the results. As expected, the sizes of the dots on the power curves show that $N = 1000$ produced the least variable power estimates. Alignments with about 250 sites therefore seem sufficient to infer EDS, while alignments with 1000 sites or more are likely to produce more precise results.

4.4.4 Number of sequences

The study undertaken in this subsection investigates the effect of the number of sequences in an alignment on the ability to predict the sites and the amino acids affected by EDS. Results were obtained from 44 simulation runs. The alignment generation and modelling procedures followed the default settings, except for the varying numbers of sequences.

Given that inferences are site-specific, their accuracy depends on the amount of information available at each site. The alignments were generated such that the number of foreground branches (= 25% of all the taxa), and consequently the number of visible substitutions toward the target residue, increases as the number of taxa increases. Therefore, larger alignments are expected to improve the quality of results, because they contain more information per site.

Figure 4.17 summarises the results obtained for this study. The pattern of the power curves agrees with the stated expectation. It shows that the ability to detect EDS increases as the number of sequences increases. There however appears to be no considerable difference among the PPV plots.



(a) Power curves (b) Positive predictive values

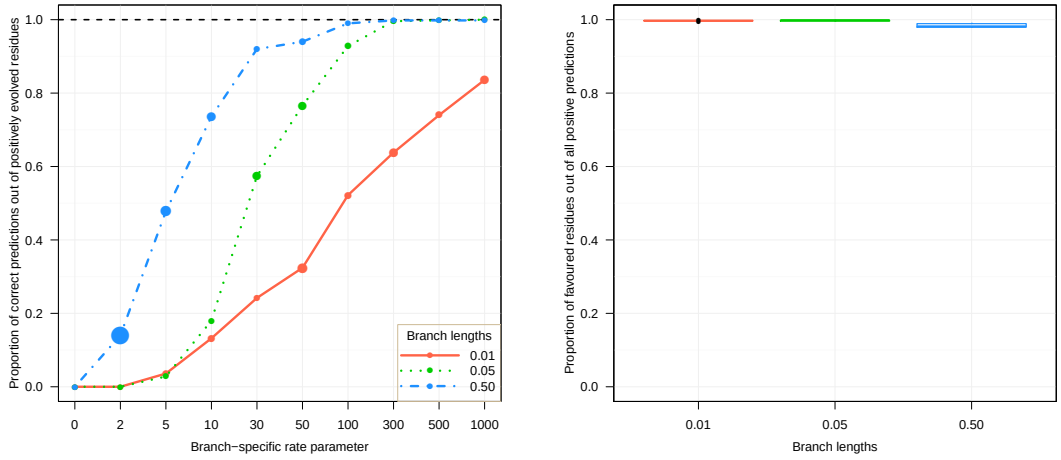
Figure 4.17: Plots obtained from analyses of alignments that differed by the number of sequences they contained. The area of the dots on the curves are proportional to the variance across simulations.

4.4.5 Branch length

This section is dedicated to investigating the effect of variation in branch lengths on EDS inference. Longer branch lengths imply longer divergence times and hence more time to accumulate information to detect EDS. Longer branch lengths also permit accumulation of random mutations that may obscure evidence about directional evolution. There are thus no clear expectations about the effects of branch length variation on the ability to infer EDS.

Default specifications were used for data generation and modelling in this subsection, except that alternative branch lengths were considered. Due to limited computational resources, only 5 simulation runs were completed. The results are summarised in Figure 4.18.

Plot 4.18(a) shows that the ability to detect EDS tends to improve as the branches become longer, while Plot 4.18(b) shows high PPVs that are insensitive to variation in branch lengths.



(a) Power curves (b) Positive predictive values

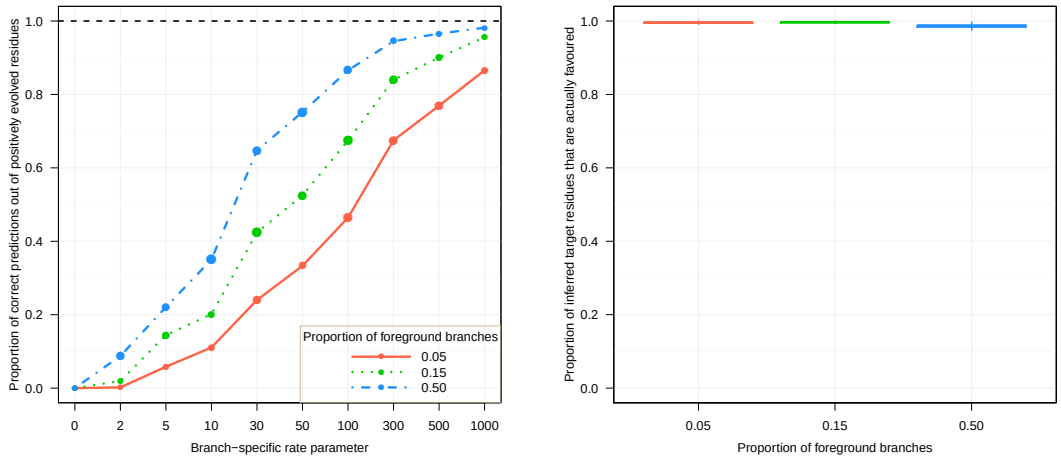
Figure 4.18: Visual representation of results from a simulation study that investigated how the lengths of the branches on a balanced phylogeny affect phylogenetic inference. The area of the dots on the curves are proportional to the variance across simulations.

4.4.6 Proportion of foreground branches

Every simulation scenario considered previously used phylogenies that were such that the foreground branches were 25% of the terminal branches. In this subsection, the effect of varying that proportion is explored. The presented results were obtained from 16 simulation runs. With the exception of the varying proportion of foreground branches all of the data generation and the modelling settings were left at their default values.

Altering the proportion of foreground branches affects the amount of information available for detecting EDS. The power to correctly infer EDS is thus expected to improve as the proportion of foreground branches increases. Figure 4.19 presents a visual summary of the results from the study.

Figure 4.19 shows results that agree with expectations. Plot 4.19(a) shows that fewer proportion of foreground branches tends to hinder the power to detect the sites and the corresponding amino acids under EDS. Plot 4.19(b) however shows that the PPVs are insensitive to the number of foreground branches.



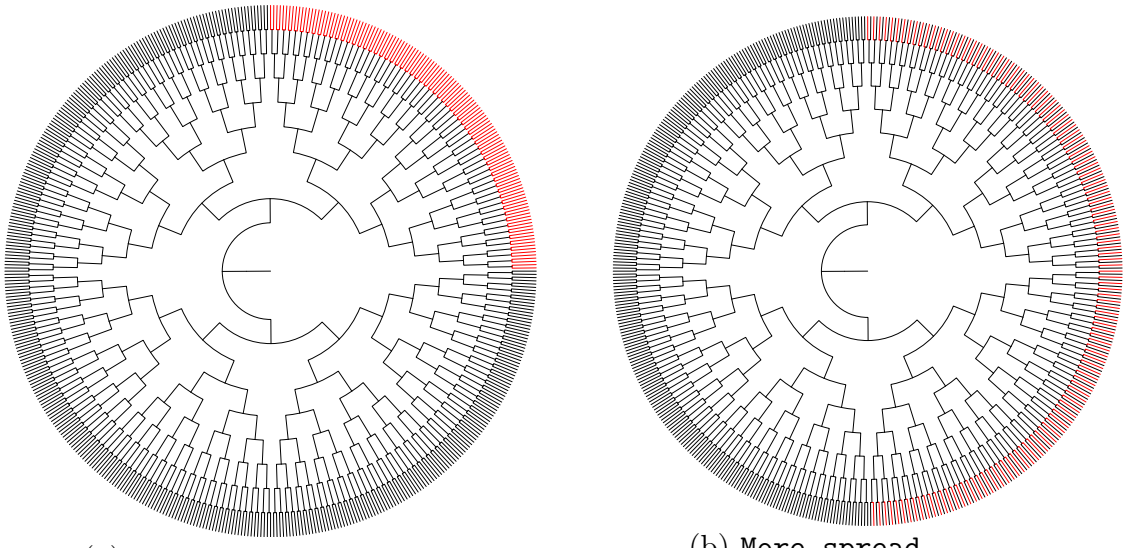
(a) Power curves (b) Positive predictive values

Figure 4.19: Visual comparison of results obtained from study designed to examine the effect the proportion of foreground branches had on episodic evolution detection. The area of the dots on the curves are proportional to the variance across simulations.

4.4.7 Distribution of foreground branches

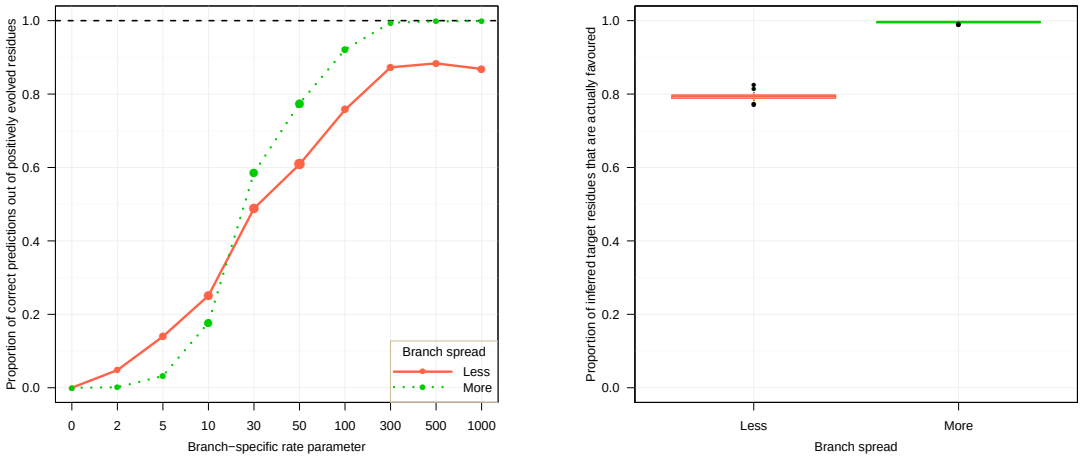
Murrell et al. (2012a) stated that the spread of the background branches across the phylogeny affects the power to detect EDS. They claimed that founder effects that tend to hinder EDS inference will be less likely the more spread out the background branches are about the phylogeny. A study designed to investigate this claim is presented in this subsection. The compared inferences are from two sets of analyses that differ only in the spread of the foreground and background branches about the phylogeny. The two types of phylogenies are shown in Figure 4.20.

In order to understand the procedure employed to construct the phylogenies in Figure 4.20, let the traversal order of the 512 terminal branches in a phylogeny be given as (`S001`, `S002`,. . . , `S512`). Assume an arbitrary random sampling technique returned `S098` from the ordered branches. Mark branch `S098` as a foreground branch. To construct the less spread phylogeny in Subfigure 4.20(a), subsequently select (`S099`, `S100`, . . . , `S225`) as foreground branches. For Subfigure 4.20(b) subsequent foreground branches need to be (`S100`, `S102`, . . . , `S352`). All the other data generation and modelling parameters were set at their default values. The results obtained from 25 simulation



(a) `Less spread` (b) `More spread`

Figure 4.20: Examples of topologies characterised by different distribution of foreground branches. Each topology contains 512 terminal taxa in total 25% (= 128) of which are selected as foreground branches, and shaded red.



(a) Power curves (b) Positive predictive values

Figure 4.21: Plots comparing the results obtained from an EDS simulation study that investigated the effect of the spread of the foreground and background branches about the phylogeny. The area of the dots on the curves are proportional to the variance across simulations.

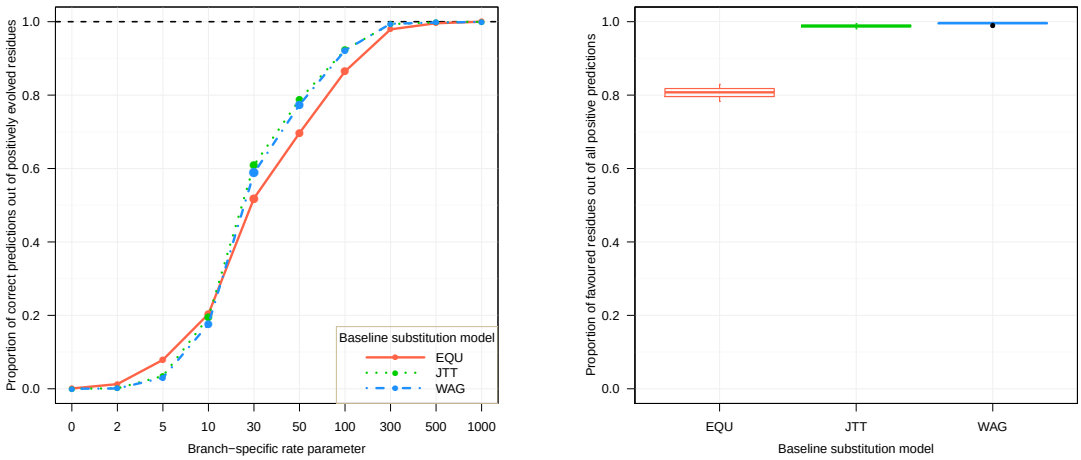
runs are visually summarised in Figure 4.21 and the results support the claim by Murrell et al. (2012a).

4.4.8 Baseline substitution model

This subsection examines the effect of using an incorrect baseline substitution model for data modelling. The effect, if any, is expected to depend on the similarity between the substitution matrix used to generate alignments and the matrix used for modelling. In conducting this study, alignments simulated with the WAG (Whelan and Goldman, 2001) substitution matrix were modelled using the true, JTT (Jones et al., 1992) and EQU (Wilbur, 1985) substitution matrices. The EQU matrix, as used here, had instantaneous substitution rates that were all equal to one and equilibrium frequencies that were all equal to 0.05. Analysis results obtained from 33 simulation runs are presented in Figure 4.22.

The power and the PPV statistics presented in Figure 4.22 show that the correct WAG matrix produced the most accurate results, which were closely followed by those produced by the JTT matrix, while the EQU matrix was generally outperformed. The three exchangeability matrices are compared in Figure 4.23. The figure shows that, as expected, results from the JTT

matrix were very similar to those from the WAG matrix, because the two matrices are similar, but different from the EQU matrix.



(a) Power curves (b) Positive predictive values

Figure 4.22: Comparative plots constructed from analysing alignments generated by WAG baseline model with the correct model, JTT and EQU models. The area of the dots on the curves are proportional to the variance across simulations.

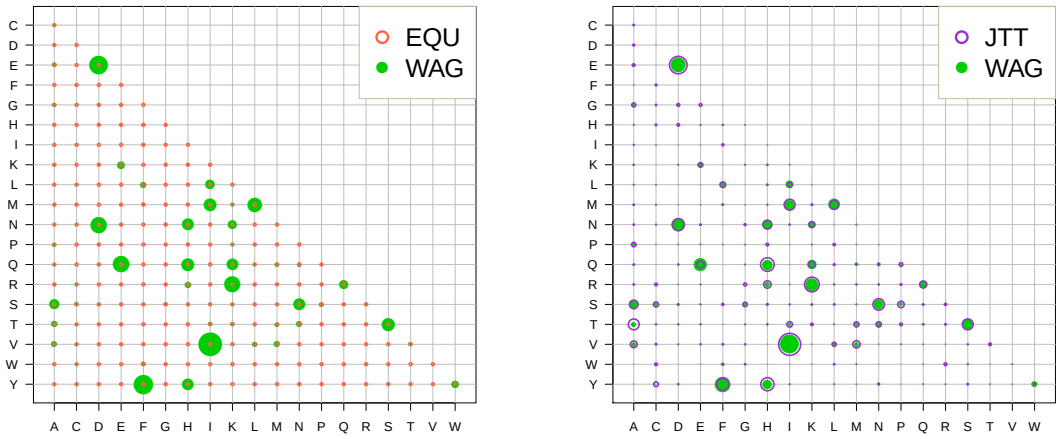


Figure 4.23: Visual illustration of how the EQU and the JTT exchangeability matrices compare to the WAG matrix. The EQU exchangeabilities are all equal to one. The sizes of the dots are relative to the corresponding exchangeabilities.

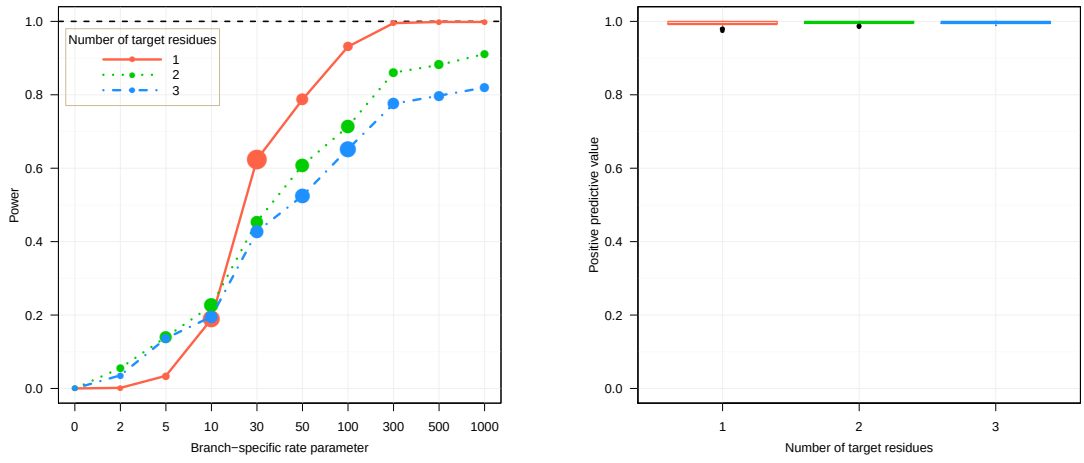
Efforts invested in identifying the most optimal baseline model for EDS analysis are repaid with more powerful and more reliable conclusions. As a result, it is recommended to model an alignment with the baseline model that best describes that alignment.

4.4.9 Number of target residues at a site

This subsection addresses how the number of the site-specific target set affects the ability of the model to detect EDS. The alignment generation and modelling procedures adopted in this subsection were as outlined in the default section, except that the number of target residues per alignment site, $||\mathbf{y}_n||$, used during the generation stage was altered. For values of $||\mathbf{y}_n||$ that were greater than one, the multiple elements of $\mathbf{y}_n$ were randomly sampled and it was ensured that no residue was sampled more than once at a particular site.

An alignment characterised by multiple target residues per site is expected to lead to poorer conclusions, because the evidence for EDS is divided between the target residues. Results were collected from 25 completed simulation runs for this study and the summaries are presented in Figure 4.24.

Figure 4.24 shows results that agree with the expectations. They show that more target residues per site led to less powerful conclusions. There is however no significant difference in the positive predictive values.



(a) Power curves (b) Positive predictive values

Figure 4.24: Illustration of how the number of target residues per site affects the ability to detect episodic directional evolution. The area of the dots on the curves are proportional to the variance across simulations.

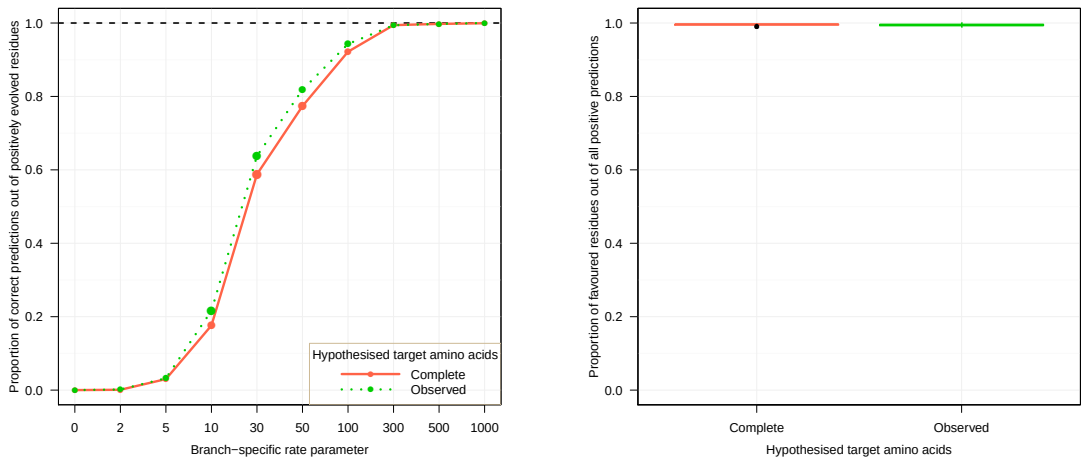
4.4.10 Set of target residues

In this subsection, the effect of the specified set of target amino acids per site is examined. In one scenario, all twenty possible amino acid residues were investigated for evidence of EDS at each site. In the other scenario, only the amino acids observed at a site were considered as targets at that particular site. No evidence in support of or against EDS is expected to be available with respect to a specific residue at a certain site, if the residue was not observed at that site (see Table 4.3 and/or Figure 4.7). Consequently, no differences are expected between the results of the analyses in this subsection.

Results were obtained from 33 simulation runs and some relevant statistics are summarised in Figure 4.25 and Table 4.9. The data generation and modelling procedures were as stipulated in the default settings and only the hypothesised set of target residues were altered as explained above.

Table 4.9: Summary of the computation time (in seconds) required to successfully implement the CVB0 inference algorithm for different sets of site-specific target residues.

Set of target residues	Min.	Quantiles			Max.	Mean	Sd.
		25%	50%	75%			
`Complete`	4521.00	5104.00	5422.00	5613.00	7626.00	5434.00	454.16
`Observed`	4266.00	4384.00	4473.00	4561.00	4716.00	4483.00	125.98



(a) Power curves (b) Positive predictive values

Figure 4.25: Illustration of how restrictions in the set of amino acid residues for each alignment site affects EDS conclusions. The `Complete` analysis refers to the case where all the 20 possible amino acids were investigated for evidence of episodic directional evolution at each site. The `Observed` scenario is one where only the residues observed at a site were considered as targets for that site. The area of the dots on the curves are proportional to the variance across simulations.

The plots in Figure 4.25 show no substantial difference between the compared analyses results, as expected. However, given the reduced computational cost associated with only considering the observed residues at each site, it is recommended that this modelling approach be adopted.

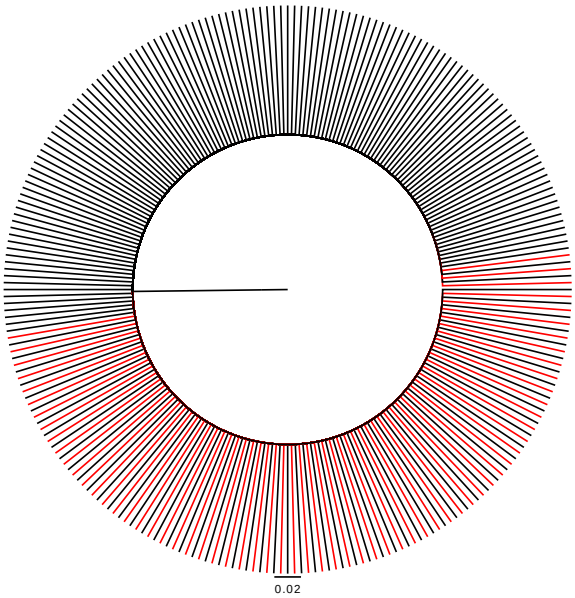
4.4.11 Phylogenetic structure

The study presented in this subsection involved investigating the effect of evolutionary tree topology on the quality of inferences. Three types of trees were analysed. They included a star tree (see Figure 4.26(a) for an illustration), an empirical tree (see Figure 4.26(b) for the tree used) and a balanced tree.

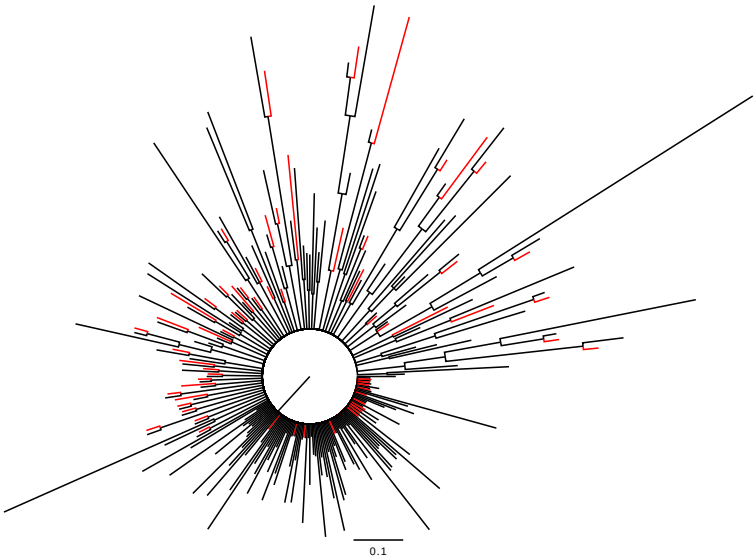
The empirical tree was obtained from an alignment of protease sequences that were obtained from Stanford HIV database (online at: `http://hivdb.stanford.edu/cgi-bin/PIPairs.cgi`). There were 303 pairs of pre- and post-treatment sequences identified whose exact collection dates were recorded and had no missing or ambiguous residues. From the identified sequences, 64 pre- and post-treatment pairs of sequences were randomly sampled. From the remaining pre-treatment data, 128 sequences were randomly selected. Consequently, a total of 256 empirical sequences were sampled such that 64 (=25%) of them were obtained post-treatment. The empirical evolutionary tree was generated as the consensus tree in `MrBayes` (Ronquist et al., 2012) from a chain length of $500,000$, a thinning size of $1,000$ and 25% burn-in proportion. The chain was considered to have converged because the reported standard deviation among chains was less than 0.1 as recommended by Ronquist et al. (2012). The terminal branches attached to the post-treatment sequences were considered as the foreground branches.

Similar to the empirical tree case, the number of extant taxa for the star and balanced trees were maintained at 256 and 64 of them were randomly chosen as the leaves of the foreground branches. Equal branch lengths of 0.0968 were used for both trees. The branch length was obtained as the mean of the empirical branch lengths.

Fifty independent sequence alignments were simulated with each of the three trees. All the alignments were subjected to episodic directional selection with parameters, except for the phylogenetic structure and number of sequences, that were set as described in Section 4.1. The false positives from the analyses were observed to be approximately zero and they were similar across all the phylogenetic structures. Power and positive predictive values from the

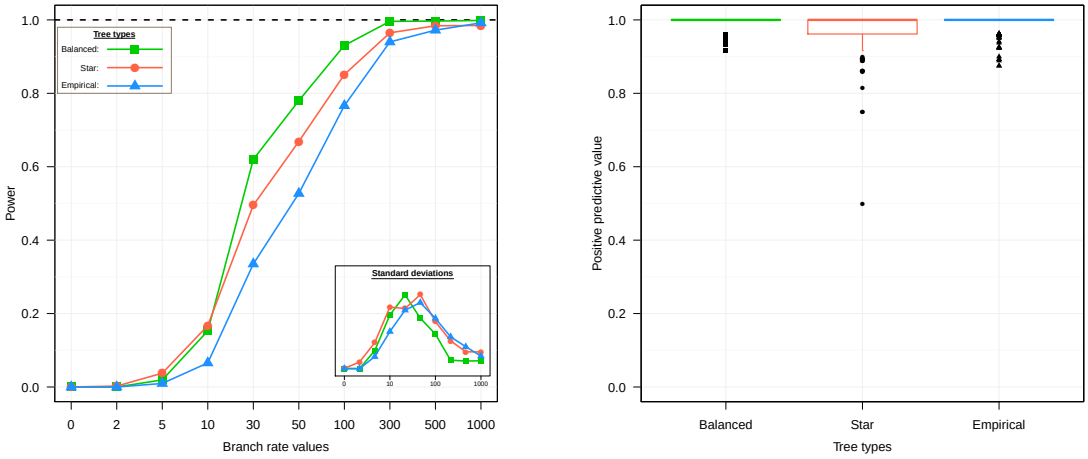


(a) Star phylogeny



(b) Empirical phylogeny

Figure 4.26: Illustration of the phylogenetic structures used during the simulation study discussed in this subsection. The red lines represent the foreground branches.



(a) Power curves (b) Positive predictive values

Figure 4.27: Power curves and positive predictive value box plots obtained from sensitivity analysis with respect to phylogenetic structure.

analysis are summarised in Figure 4.27. Except for the apparent discrepancy in the power curves for bias parameter values between 10 and 100, the accuracy of inference was unaffected by phylogenetic structure.

The sensitivity of the results from EDS analysis based on the CVB0 algorithm to variation in different model components has been successfully demonstrated in this section. Unlike the scenarios considered here, where only a specific parameter is allowed to vary for a particular analysis, empirical cases exhibit variation in combinations of the model components. The focus in the next chapter is to apply the CVB0 algorithm on some empirical HIV alignments. The recommendations from this section are duly adopted in the process.

Chapter 5

Empirical study

Murrell et al. (2012a) applied their model of episodic directional evolution of protein sequences (EDEPS) to three HIV-1 datasets. The datasets include reverse transcriptase, protease and integrase. They sought to detect sites and the corresponding amino acids that impart drug resistance. The same datasets are analysed here. The datasets and their corresponding topologies were retrieved from the MEDS section of the `HyPhy` wiki (online at: `www.hyphy.org` on January 10, 2015). In the EDEPS study, all results were compared to the list of drug resistance associated mutations (DRAMs) obtained from HIVdb [`http://hivdb.standford.edu`]. This approach was adopted here too.

In this chapter, the zero-order collapsed variational Bayes (CVB0) algorithm was applied to each of the HIV-1 alignments. The detected target amino acids that are supported by the EDEPS study and the latest DRAMs list (Wensing et al., 2014) are indicated. The DRAMs list by Wensing et al. (2014) was based on additional evidence not available to the methods considered in this dissertation. Consequently, Wensing et al. (2014) is considered as the gold standard such that the results from other models were compared to it.

5.1 Datasets

Reverse transcriptase. The reverse transcriptase dataset comprised of pairs of HIV-1 reverse transcriptase isolates collected from 238 patients before and after initiation of highly active anti-retroviral therapy. After removing sites that were difficult to align, the alignment contained 211 amino acid sites corresponding to positions 40 to 250 from the retrieved alignment. Every terminal branch leading to a post-therapy sequence was selected as

a foreground branch. The tree topology inferred by Murrell et al. (2012a) associated with this alignment is given in Figure 5.1 with the foreground branches shaded in red.

Protease. Initially, 49 protease sequences were taken from 37 patients that had been treated with protease inhibitors. Since pre-treatment protein sequences were not available, sequences from drug naive patients were obtained from the Stanford HIV database (HIVdb). Each post-treatment sequence was then matched by p-distance with a closely related pre-treatment sequence from a drug naive patient. The final dataset consisted of 122 sequences with 99 amino acid positions. All of the terminal branches leading to post-therapy sequences were selected as foreground branches. When two branches, which

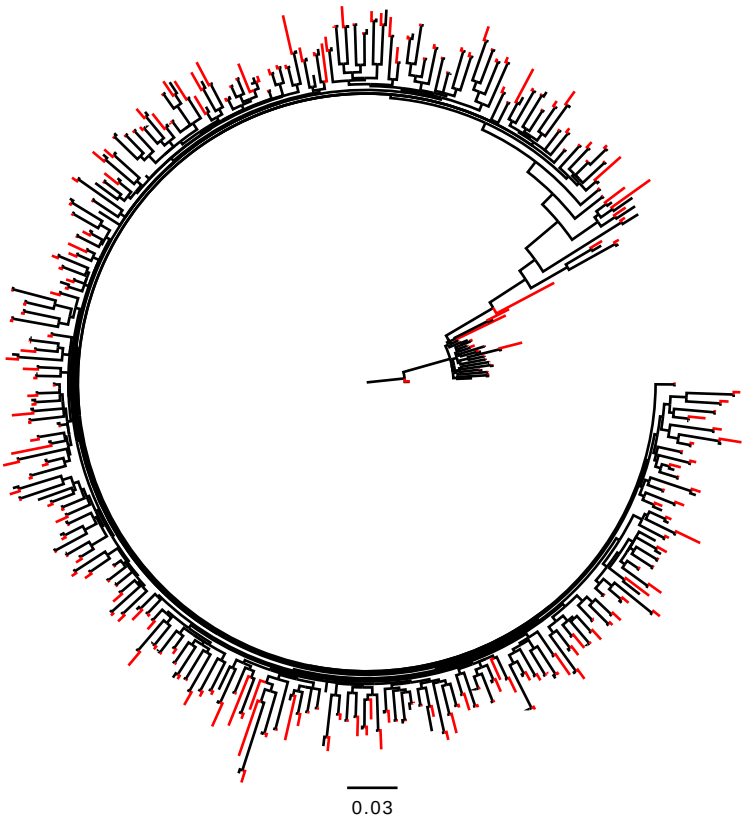


Figure 5.1: The tree topology that describes the evolutionary relationships among the sequences in the HIV-1 reverse transcriptase alignment, as supplied by Murrell et al. (2012a). The foreground branches are shaded in red.

share a common ancestor were selected as foreground branches, their ancestral branch was also selected as a foreground branch as it was done in Murrell et al. (2012a). The evolutionary tree topology for the protease alignment is shown in Figure 5.2.

Integrase. The integrase alignment comprised of 295 sequences with 288 sites. The alignment was constructed from 83 unique post-therapy sequences that were obtained from 40 patients. The drug naive sequences were obtained from the HIVdb by first matching each of the post-therapy sequences to the 25 closest pre-treatment sequences. Sequence closeness was measured by the HKY85 distance (Massingham and Goldman, 2005). The final pre-treatment sequences comprised of the unique sequences from all the identified close sequences. See Figure 5.3 for the evolutionary tree topology provided with

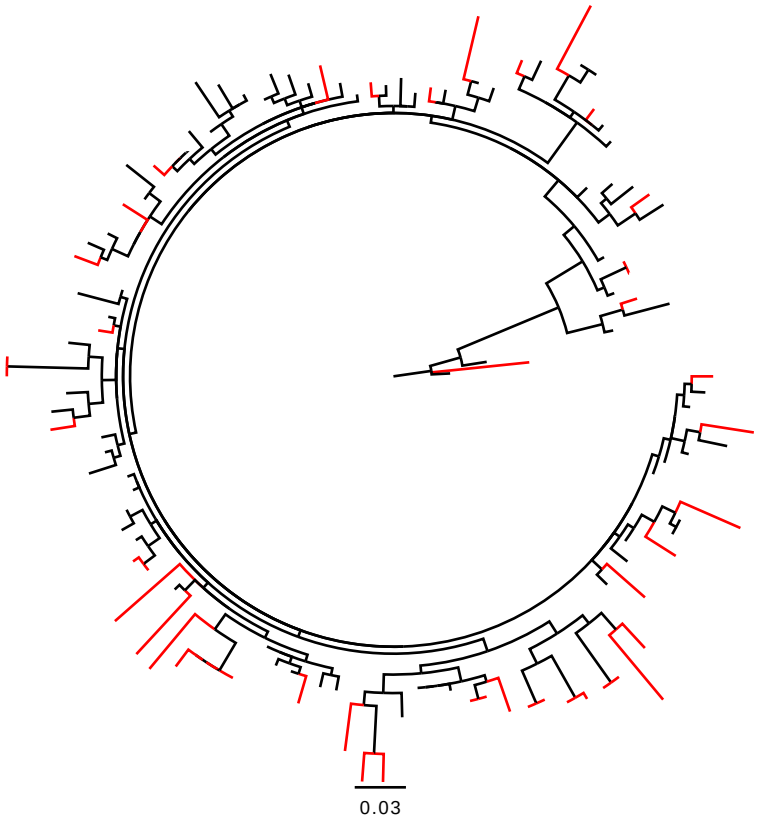


Figure 5.2: Evolutionary tree topology illustrating the relationships among the sequences contained in the analysed protease alignment. The topology is as used in the EDEPS study (Murrell et al., 2012a) and the red branches correspond to those of the foreground branches.

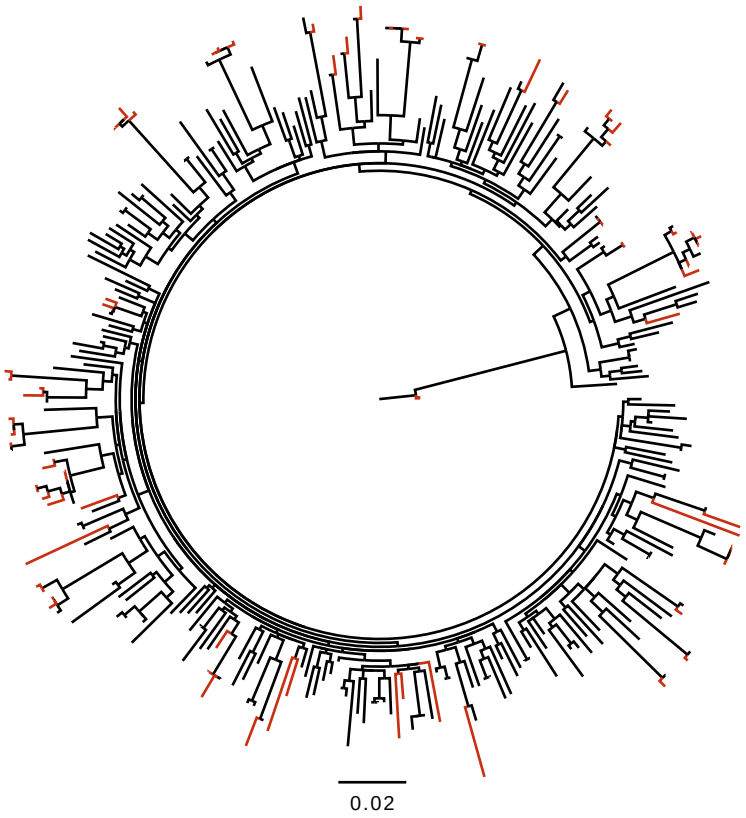


Figure 5.3: Evolutionary relationships among the integrase sequences. The topology is as used in the empirical study presented in EDEPS (Murrell et al., 2012a). The foreground branches are shaded in red.

the alignment.

5.2 Model fitting

The results from analyses of the HIV-1 datasets with the CVB0 technique are presented in this section. The same modelling procedures were followed for the three genes. The equilibrium frequencies were obtained from the corresponding alignment. Branch length estimates were obtained using a model without rate variation between sites and across branches. After obtaining estimates for the equilibrium frequencies and the branch lengths, all invariant sites were excluded as was done in the EDEPS study. The amino acids considered as potential target residues at each site were restricted to those

observed at the corresponding site. The number of pre-defined multiplier and bias rate categories was set to 20 each. The aim was to detect the sites and the corresponding amino acids under positive EDS. An amino acid at a particular site is inferred to be under positive EDS if the corresponding Bayes factor was at least 100.

Reverse transcriptase. Table 5.1 summarises the results from analysis of the reverse transcriptase alignment. The table presents the amino acids and the corresponding sites detected by the CVB0 algorithm. The table also indicates which of the inferences are supported by the EDEPS study and Wensing et al. (2014). The analyses in Murrell et al. (2012a) included the use of a codon-based model of episodic directional selection (MEDS). The predictions from the MEDS model that agree with the CVB0 approach are also shown in the table.

Table 5.1 presents the eighteen substitutions at sixteen sites detected by the CVB0 algorithm. Of these, only two (i.e. `178I` and `245V`) are not known to confer drug resistance and might be worth investigating in future study.

Wensing et al. listed a total of 55 DRAMs with respect to the reverse transcriptase gene. The EDEPS model detected 16 DRAMs while 20 DRAMs were detected in the MEDS study. All of the 9 DRAMs that the three works agree on were detected by CVB0, except for `215F`, and are listed in Table 5.1. A list of the DRAMs that were detected by EDEPS or MEDS but not EDEPS is presented in Table 5.2.

The DRAMs identified by Wensing et al. only and EDEPS that were not detected by CVB0 were `62V`, `77L` and `115F`. MEDS and Wensing et al. identified `41L` and `116Y` but this finding was not supported by CVB0 or EDEPS. Likewise, `165L` and `228R` were predicted by MEDS and EDEPS, but were unsupported by CVB0 or Wensing et al.

Only EDEPS detected `174R`. MEDS was the only analysis that detected `64K`, `98S`, `104Y`, `151Q`, `188Y` and `215T`. Of the 35 DRAMS solely reported by Wensing et al., the listed target amino acids for `65E`, `65N`, `70E`, `98G`, `101H`, `101P`, `138G`, `138K`, `138Q`, `179F`, `179L`, `179T`, `181V`, `188C`, `227C`, and `230I` were not observed in the affected sites, and could therefore not be detected by any of the algorithms given the alignment. The DRAMs supported by only Wensing et al. that correspond to target residues that were observed at the affected sites include `67N`, `74V`, `75I`, `90I`, `103S`, `106A`, `106I`, `106M`, `108I`, `138A`, `138R`, `179D`, `184I`, `188H`, `219E`, `219Q`, `210W`, `215Y` and `225H`.

Table 5.1: Sites and corresponding amino acids identified in the reverse transcriptase gene to be associated with drug resistant mutations by the CVB0 algorithm. Tick marks are used to indicate which of the predictions are supported by Wensing et al. (2014).

Site	Target	CVB0 Bayers factor	Wensing et al.	EDEPS Bayers factor	MEDS p-value
65	R	6,807	✔		
70	R	255	✔		
100	I	$> 10^5$	✔	$> 10^5$	< 0.0001
101	E	117	✔		
103	N	$> 10^5$	✔	$> 10^5$	< 0.0001
151	M	$> 10^5$	✔	$> 10^5$	< 0.0001
162	S	65,334		$1,772$	
178	I	103			
181	C	$> 10^5$	✔		
181	I	$> 10^5$	✔	$> 10^5$	< 0.0001
184	V	$> 10^5$	✔	$> 10^5$	< 0.0001
188	L	$> 10^5$	✔	$> 10^5$	< 0.0001
190	A	84,858	✔		
190	S	$> 10^5$	✔	$> 10^5$	< 0.0001
221	Y	103	✔		
228	R	15,902		0.0003	$1,401$
230	L	133	✔	0.0003	$> 10^5$
245	V	326			

The CVB0 algorithm produced Bayes factors of at least three for 11 of the 26 DRAMs that it did not detect but were detected by at least one of Wensing et al., the EDEPS and MEDS. This is noteworthy because Kass and Raftery (1995) recommended three as the Bayes factor threshold for inconclusive evidence.

Table 5.2: HIV-1 reverse transcriptase DRAMs that were detected by MEDS or EDEPS but not CVB0. Only p-values ≤ 0.01 (after adjusting for multiple comparisons) and Bayes factors > 100 were reported in Murrell et al. (2012a). The sites analysed in this work were from 40 to 250, any sites not in that range were marked with `"*"` under CVB0.

Site	Target	CVB0 Bayers factor	Wensing et al.	EDEPS Bayers factor	MEDS p-value
41	L	16.572	✔	-	0.0026
62	V	34.416	✔	313	-
64	K	1.953		-	0.0024
77	L	5.722	✔	211	-
98	S	9.855		-	0.0049
104	Y	6.734		-	0.0024
115	F	2.815	✔	3,110	-
116	Y	20.983	✔	-	0.0032
151	Q	0.496		-	0.0002
165	L	69.831		2,245	< 0.0001
174	R	8.392		105	-
188	Y	0.330		-	< 0.0001
215	F	3.496	✔	2,727	0.0028
215	T	0.841		-	0.0004
286	A	*		-	0.0009

Protease The results in Table 5.3 correspond to the analogue of Table 5.1 for the protease gene. The table shows that all of the four drug resistant mutations detected by CVB0 are supported by the EDEPS or the MEDS study by Murrell et al. (2012a) or by the clinically verified DRAMs listed in Wensing et al. (2014).

In total, Wensing et al. listed 75 protease DRAMs and 37 of them are associated with target amino acids that were not observed at the affected sites in the analysed alignment. The EDEPS study detected 5 DRAMs. MEDS analysis predicted 9 DRAMs. Only `84V` and `90M` were jointly supported by the results from these three analyses and the CVB0 algorithm also detected the latter.

Wensing et al. and the EDEPS agree that `71V` and `82A` are DRAMs in the protease gene. Wensing et al. and MEDS additionally identified `60E` and

Table 5.3: Sites and corresponding amino acids detected in the protease gene to be associated with drug resistance mutations by the CVB0 algorithm. Tick marks are used to indicate which of the detections are supported by the EDEPS and the MEDS study and Wensing et al. (2014). The values under the CVB0 column are the associated Bayes factors.

Site	Target	CVB0 Bayers factor	Wensing et al.	EDEPS Bayers factor	MEDS p-value
12	T	102			< 0.0001
20	R	224	✔		
82	A	$> 10^5$	✔	$> 10^5$	
90	M	$> 10^5$	✔	$> 10^5$	< 0.0001

93L. Both the EDEPS and the MEDS analyses detected 13V. The 12T, 35D, 61E and 74S DRAMs were only detected by MEDS. The results summarised in Table 5.4 are all the cases observed with Bayes factors less than 100, after CVB0 analysis of the protease gene, which were identified as DRAMs by EDEPS or MEDS.

Integrase After analysing the integrase gene with the CVB0 algorithm, six drug resistance associated mutations were detected. All of the detected mutations were supported by the clinically verified DRAMs listed by Wensing et al. (2014) or experimentally detected DRAMs by Murrell et al. (2012a). The results are summarised in Table 5.5 in a format similar to that of tables 5.1 and 5.3.

Table 5.4: HIV-1 protease DRAMs were detected by MEDS or EDEPS but not CVB0. Only p-values ≤ 0.01 (after adjusting for multiple comparisons) and Bayes factors > 100 were reported in Murrell et al. (2012a).

Site	Target	CVB0 Bayers factor	Wensing et al.	EDEPS Bayers factor	MEDS p-value
13	V	7.185	✔	145	0.0059
35	D	1.676		-	0.0035
60	E	2.756	✔	-	< 0.0001
61	E	34.425	✔	-	< 0.0001
71	V	3.577	✔	257	-
74	S	2.224	✔	-	0.0007
84	V	6.124	✔	$> 10^5$	0.0080
93	L	18.030	✔	-	0.0078

Table 5.5: Drug resistant associated mutations detected by the CVB0 algorithm from analysis of the integrase gene. The table shows the detected target sites and the corresponding amino acids as well as the associated Bayes factors for the CVB0 approach. The mutations supported by the EDEPS and the MEDS studies as well as Wensing et al. (2014) are indicated with tick marks.

Site	Target	CVB0 Bayers factor	Wensing et al.	EDEPS Bayers factor	MEDS p-value
74	M	1,443	✔		
97	A	2,552	✔	$> 10^5$	0.0028
140	S	$> 10^5$	✔	$> 10^5$	< 0.0001
143	R	1,757	✔	$> 10^5$	0.0015
148	H	72,986	✔	$> 10^5$	< 0.0001
155	H	$> 10^5$	✔	$> 10^5$	< 0.0001

Twenty DRAMs were listed by Wensing et al. for the integrase gene. Eight of these, namely 66A, 66I, 66K, 121Y, 138K, 140A, 147G and 148K, could not be detected by the CVB0 algorithm because the target residues involved were not observed at the corresponding sites in the analysed alignment. Results from analyses performed with MEDS, EDEPS and the CVB0 algorithms jointly agree with five of the DRAMs listed by Wensing et al. (see Table 5.5). Contrarily, 92G, 92Q, 138A, 143C, 143H and 148R were listed by Wensing et al. but were not detected by any of the CVB0, EDEPS or MEDS algorithms. Only the EDEPS study detected evidence of EDS at positions 163R and 221Q. Similarly, only the MEDS implementation detected 72I. The DRAMs detected by EDEPS or MEDS but not CVB0 are summarised in Table 5.6.

Table 5.6: HIV-1 integrase DRAMs were detected by MEDS or EDEPS but not CVB0. Only p-values ≤ 0.01 (after adjusting for multiple comparisons) and Bayes factors > 100 were reported in Murrell et al. (2012a).

Site	Target	CVB0 Bayers factor	Wensing et al.	EDEPS Bayers factor	MEDS p-value
72	I	4.733	✔	-	0.0095
163	R	13.152	✔	1,143	-
221	Q	1.220		107	-

Chapter 6

Conclusions

The phylogenetic problem addressed in this minor dissertation concerns detecting protein alignment sites and the corresponding amino acids targeted by episodic directional selection (EDS). This problem was identified as analogous to the information retrieval issue, of uncovering the hidden topics that characterise an observed document of words, addressed by the latent Dirichlet allocation (LDA) (Blei et al., 2003). Therefore, it was possible to use the LDA inference routines in an EDS context for the first time.

Three statistical techniques for solving the EDS problem were considered. These were the collapsed Gibbs sampling (CGS), the collapsed variational Bayes (CVB) and the zero-order collapsed variational Bayes (CVB0) algorithms. They combined ideas from the LDA models and the fast unconstrained Bayesian approximation model of Murrell et al. (2013). These techniques could contribute toward solving the HIV pandemic by detecting new drug resistance patterns. Despite the focus of this research on HIV-1, the presented algorithms are directly applicable to other pathogens.

The performances of the deterministic CVB and CVB0 techniques on alignments generated under various circumstances were investigated using the stochastic CGS algorithm as the benchmark. The CVB0 algorithm produced results that were very similar to those of the CGS algorithm, but was computationally much more efficient than CGS and CVB. Consequently, the CVB0 algorithm was applied to detect sites evolving under EDS in three HIV-1 alignments that contained sequences taken before and after antiretroviral therapy. The results from this empirical study were largely supported by previous studies. Two novel mutations, namely 178I and 245V, were detected in the reverse transcriptase gene. It is recommended that these mutations be investigated as potential contributors to drug resistance.

Future work could include the development of a phylogenetic model of codon

evolution. It might also be interesting to relax the assumption that the foreground and background partitions of the branches in the phylogeny are known. Each branch could then be probabilistically classified into the two branch categories.

Bibliography

Anisimova, M. and Yang, Z. (2007). Multiple Hypothesis Testing to Detect Lineages under Positive Selection that Affects Only a Few Sites, *Molecular Biology and Evolution* **24**(5): 1219–1228. doi:10.1093/molbev/msm042.

Asuncion, A. U. (2010). Approximate Mean Field for Dirichlet-Based Models, *27th International Conference of Machine Learning*, Haifa, Israel.

Asuncion, A., Welling, M., Smyth, P. and Teh, Y. W. (2009). On Smoothing and Inference for Topic Models, *Uncertainty in Artificial Intelligence*.

Bishop, C. M. (ed.) (2007). *Pattern Recognition and Machine Learning*, Springer Science+Business Media, LLC.

Blanquart, S. and Lartillot, N. (2008). A Site- and Time-Heterogeneous Model of Amino Acid Replacement, *Molecular Biology and Evolution* **25**(5): 842–858. doi:10.1093/molbev/msm018.

Blei, D. M., Ng, A. Y. and Jordan, M. I. (2003). Latent Dirichlet Allocation, *Journal of Machine Learning Research* **3**: 993–1022.

Brooks, S., Gelman, A., Jones, G. I. and Meng, X.-L. (eds) (2011). *Handbook of Markov Chain Monte Carlo*, Chapman and Hall/CRC.

Cao, Y., Adachi, J., Janke, A., Pääbo, S. and Hasegawa, M. (1994). Phylogenetic Relationships Among Eutherian Orders Estimated from Inferred Sequences of Mitochondrial Proteins: Instability of a Tree Based on a Single Gene, *Journal of Molecular Evolution* **39**(5): 519–527.

Cavalli-Sforza, L. L. and Edwards, W. F. (1967). Phylogenetic Analysis Models and Estimation Procedures, *American Journal of Human Genetics* **19**(3): 233–257.

Clavel, F. and Hance, A. J. (2004). HIV Drug Resistance, *Journal of Medicine* **350**(10): 1023–1035.

Darling, W. M. (2011). A Theoretical and Practical Implementation Tutorial on Topic Modelling and Gibbs Sampling, *Technical report*, School of Computer Science, University of Guelph.

Dempster, A. P., Laird, N. M. and Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm, *Journal of the Royal Statistical Society* **39**(1): 1–38.

Felsenstein, J. (1973). Maximum Likelihood and Minimum-steps Methods of Estimating Evolutionary Trees from Data on Discrete Characters, *Systematic Zoology* **22**(3): 240–249.

Felsenstein, J. (1981). Evolutionary Trees from DNA Sequences: a Maximum Likelihood Approach, *Molecular Biology and Evolution* **17**: 368–376.

Felsenstein, J. (2004). *Inferring Phylogenies*, Sinauer Associates, Inc.

Foster, P. G. (2004). Modelling Compositional Heterogeneity, *Systematic Biology* **53**(3): 485–495. DOI:10.1080/10635150490445779.

Geweke, J. (1992). Evaluating the Accuracy of Sampling-Based Approaches to the Calculation of Posterior Moments, *IN BAYESIAN STATISTICS*, University Press, pp. 169–193.

Goldman, N., Thorne, J. L. and Jones, D. T. (1998). Assessing the Impact of Secondary Structure and Solvent Accessibility on Protein Evolution, *Genetics* **149**(1): 445–458.

Griffiths, T. L. and Steyvers, M. (2004). Finding scientific topics, *PNAS* **101**(1): 5228–5235. doi:/10.1073/pnas.0307752101.

Hamilton, N. (2014). *ggtern: An extension to ggplot2, for the creation of ternary diagrams.* R package version 1.0.3.2.
URL: *http://CRAN.R-project.org/package=ggtern*

Hofmann, T. (2001). Unsupervised Learning by Probabilistic Latent Semantic Analysis, *Machine Learning* **42**: 177–196.

Jia, F., Lo, N. and Ho, S. Y. W. (2014). The Impact of Modelling Rate Heterogeneity among Sites on Phylogenetic Estimates of Intraspecific Evolutionary Rates and Timescales, *PLoS ONE* **9**(5): e95722. doi:10.1371/journal.pone.0095722.

Jones, D. T., Taylor, W. R. and Thorton, J. M. (1992). The Rapid Generation of Mutation Data Matrices from Protein Sequences, *Computer Application in the Biosciences* **8**(3): 275–282. doi:10.1093/bioinformatics/8.3.275.

Kass, R. E. and Raftery, A. E. (1995). Bayes Factors, *Journal of the American Statistical Association* **90**(430): 773–795.

Kosakovsky Pond, S. L., Delport, W., Muse, S. V. and Scheffler, K. (2010). Correcting the Bias of Empirical Frequency Parameter Estimators in Codon Models, *PLoS ONE* **5**(7): e11230. DOI: 10.1371/journal.pone.0011230.

Kosakovsky Pond, S. L. and Frost, S. D. W. (2005a). A Genetic Algorithm Approach to Detecting Lineage-Specific Variation in Selection Pressure, *Molecular Biology and Evolution* **22**(3): 478–485. doi:10.1093/molbev/msi031.

Kosakovsky Pond, S. L. and Frost, S. D. W. (2005b). Not so different after all: a comparison of methods for detecting amino acid sites under selection, *Molecular Biology and Evolution* **22**(5): 1208–1222. doi:10.1093/molbev/msi105.

Kosakovsky Pond, S. L., Frost, S. D. W. and Muse, S. V. (2005). HyPhy: Hypothesis Testing Using Phylogenies, *Bioinformatics* **21**(5): 676–679. doi:10.1093/bioinformatics/bti079.

Kosakovsky Pond, S. L., Murrell, B., Fourment, M., Frost, S. D. W., Delport, W. and Scheffler, K. (2011). A Random Effects Branch-Site Model for Detecting Episodic Diversifying Selection, *Molecular Biology and Evolution* **28**(11). doi:10.1093/molbev/msr125.

Kosakovsky Pond, S. L., Poon, A. F. Y., Brown, A. J. L. and Frost, S. D. W. (2008). A Maximum Likelihood Method for Detecting Directional Evolution in Protein Sequences and Its Application to Infiuenza A Virus, *Molecular Biology and Evolution* **25**(9): 1809–1824. doi:10.1093/molbev/msn123.

Kosakovsky Pond, S. L., Poon, A. F. Y. and Frost, S. D. W. (2009). Estimating Selection Pressures on Alignments of Coding Sequences, *in* P. Lemey, M. Salemi and A.-M. Vandamme (eds), *The Phylogenetic Handbook: a Practical Approach to Phylogenetic Analysis and Hypothesis Testing*, Cambridge University Press, chapter 14, pp. 419–451.

Kullback, S. and Leibler, R. A. (1951). On Information and Sufficiency, *The Annals of Mathematical Statistics* **22**(1): 79–86.

Lacerda, M., Scheffler, K. and Seoighe, C. (2010). Epitope Discovery with Phylogenetic Hidden Markov Models, *Molecular Biology and Evolution* **27**(5): 1212–1220. doi:10.1093/molbev/msq008.

Liò, P. and Goldman, N. (1998). Models of Molecular Evolution and Phylogeny, *Genome Research* **8**: 1233–1244. doi:10.1101/gr.8.12.1233.

Massingham, T. and Goldman, N. (2005). Detecting Amino Acid Sites Under Positive Selection and Purifying Selection, *Genetics* **169**(3): 1753–1762. doi:10.1534/genetics.104.032144.

McDonald, J. H. and Kreitman, M. (1991). Adaptive Protein Evolution at the Adh locus in Drosophila, *Nature* **351**(6328): 652–654. doi:10.1038/351652a0.

Murrell, B., Moola, S., Mabona, A., Weighill, T., Sheward, D., Kosakovsky Pond, S. L. and Scheffler, K. (2013). FUBAR: a fast, unconstrained Bayesian approximation for inferring selection, *Molecular Biology and Evolution* **30**(5): 1196–1205. doi:10.1093/molbev/mst030.

Murrell, B., Oliveira, T., Seebregts, C., Kosakovsky Pond, S. L., Scheffler, K. and on behalf of the Southern African Treatment and Resistance Network SATuRN Consortium (2012a). Modeling HIV-1 Drug Resistance as Episodic Directional Selection, *PLoS Computational Biology* **8**(5). e1002507. doi:10.1371/journal.pcbi.1002507.

Murrell, B., Wertheim, J. O., Moola, S., Weighill, T., Scheffler, K. and Kosakovsky Pond, S. L. (2012). Detecting Individual Sites Subject to Episodic Diversifying Selection, *PLoS Genetics* **8**(7). doi:10.1371/journal.pgen.1002764.

Nickle, D. C., Heath, L., Jensen, M. A., Gilbert, P. B., Mullins, J. I. and Kosakovsky Pond, S. L. (2007). HIV-Specific Probabilistic Models of Protein Evolution, *PLoS ONE* **2**(6): e503. doi:10.1371/journal.pone.0000503.

Nielsen, R. and Yang, Z. (1998). Likelihood Models for Detecting Positively Selected Amino Acid Sites and Applications to the HIV-1 Envelope Gene, *Genetics* **148**: 929–936.

Novak, R. M., Chen, L., MacArthur, R. D., Baxter, J. D., Hullsiek, K. H., Peng, G., Xiang, Y., Henely, C., Schmetter, B., Uy, J., van den Berg-Wolf, M., Kozal, M. and the Terry Beirn Community Programs for Clinical Research on AIDS 058 Study Team (2005). Prevalence of Antiretroviral Drug Resistance Mutations in Chronically HIV-Infected, Treatment-Naive Patients: Implications for Routine Resistance Screening before Initiation of Antiretroviral Therapy, *Clinical Infectious Diseases* **40**: 468–474.

Plummer, M., Best, N., Cowles, K. and Vines, K. (2006). CODA: Convergence Diagnosis and Output Analysis for MCMC, *R News* **6**(1): 7–11.
URL: *http://CRAN.R-project.org/doc/Rnews/*

Quang, S., Lartillot, N. and Gasquet, O. (2008). Phylogenetic mixture models for proteins, *Philosophical Transactions of The Royal Society* **363**: 3965–3976. doi:10.1098/rstb.2008.0180.

R Core Team (2013). *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.
URL: *http://www.R-project.org/*

Ronquist, F., Teslenko, M., Mark, P. V. D., Ayres, D. L., Darling, A., Höhna, S., Larget, B., Liu, L., Suchard, M. A. and Huelsenbeck, J. P. (2012). MrBayes 3.2: Efficient bayesian phylogenetic inference and model choice across a large model space, *Systematic Biology* **61**: 1–4. DOI:10.1093/sysbio/sys029.

Ross, S. M. (2010). *Introduction to Probability Models*, 10th edn, Elseiver Inc.

Sato, I. and Nakagawa, H. (2012). Rethinking Collapsed Variational Bayes Inference for LDA, *Proceedings of the 29th International Conference on Machine Learning*.

Sorhannus, U. and Kosakovsky Pond, S. L. (2006). Evidence for Positive Selection on a Sexual Reproduction Gene in the Diatom Genus Thalassiosira (Bacillariophyta), *Journal of Molecular Evolution* **63**: 231–239. DOI: 10.1007/s00239.006-0016-z.

Stark, A., Lin, M. F., Kharadpour, P., Pedersen, J. S., Parts, L., Carlson, J. W., Crosby, M. A., Rasmussen, M. D., Roy, S., Deoras, A. N., Ruby, J. G., Brennecke, J., FiyBase Curators, H., project, B. D. G., Hodges, E., Hinrichs, A. S., Caspi, A., Paten, B., Park, S.-W., Han, M. V., Maeder,

M. L., Polansky, B. J., Robson, B. E., Aerts, S., van Helden, J., Hassan, B., Gilbert, D. G., Eastman, D. A., Rice, M., Weir, M., Hahn, M. W., Park, Y., Dewey, C. N., Pachter, L., Kent, W. J., Haussler, D., Lai, E. C., Bartel, D. P., Hannon, G. J., Kaufman, T. C., Eisen, M. B., Clark, A. G., Smith, D., Celniker, S. E., Gelbart, W. M. and Kellis, M. (2007). Discovery of functional elements in 12 Drosophila genomes using evolutionary signatures, *Nature* **450**(8): 219–232. doi:10.1038/nature06340.

Teh, Y. W., Newman, D. and Welling, M. (2007). A Collapsed Variational Bayesian Inference Algorithm for Latent Dirichlet Allocation, *Advances in Neural Information Processing Systems* **19**(1353).

Warnes, G. R., Bolker, B. and Lumley, T. (2014). *gtools: Various R programming tools*. R package version 3.4.1.
URL: *http://CRAN.R-project.org/package=gtools*

Wensing, A. M., Calvez, V., Günthard, H. F., Johnson, V. A., Paredes, R., Pillay, D., Shafer, R. W. and Richman, D. D. (2014). Special Contribution: 2014 Update of the Drug Resistance Mutations in HIV-1, *Topics in Antiviral Medicine* **22**(3): 642–650.

Whelan, S. and Goldman, N. (2001). A General Empirical Model of Protein Evolution Derived from Multiple Protein Families Using a Maximum-Likelihood Approach, *Molecular Biology and Evolution* **18**(5): 691–699.

Wilbur, W. J. (1985). On the PAM Matrix Model of Protein Evolution, *Molecular Biology and Evolution* **2**(5): 434–447.

Yang, Z. (1994). Estimating the Pattern of Nucleotide Substitution, *Journal of Molecular Evolution* **39**: 105–111.

Yang, Z. (1997). PAML: a program package for phylogenetic analysis by maximum likelihood, *CABIOS APPLICATION NOTE* **13**(5): 555–556.

Yang, Z. (1998). Likelihood Ratio Tests for Detecting Positive Selection and Application to Primate Lysozyme Evolution, *Molecular Biology and Evolution* **15**(5): 568–573.

Yang, Z. (2006). *Computational Molecular Evolution*, Oxford University Press.

Yang, Z. and Nielsen, R. (2002). Codon-Substitution Models for Detecting Molecular Adaptation at Individual Sites Along Specific Lineages, *Molecular Biology and Evolution* **19**(6): 908–917.

Yang, Z., Nielsen, R., Goldman, N. and Pedersen, A.-M. K. (2000). Codon-Substitution Models for Heterogeneous Selection Pressure at Amino Acid Sites, *Genetics* **155**: 431–449.

Yang, Z. and Swanson, W. J. (2002). Codon-Substitution Models to Detect Adaptive Evolution that Account for Heterogeneous Selective Pressures Among Site Classes, *Molecular Biology and Evolution* **19**(1): 49–57.

Zhang, J. (2004). Frequent False Detection of Positive Selection by the Likelihood Method with Branch-Site Models, *Molecular Biology and Evolution* **21**(7): 1332–1339. doi: 10.1093/molbev/msh117.

Appendix A

Dirichlet distribution

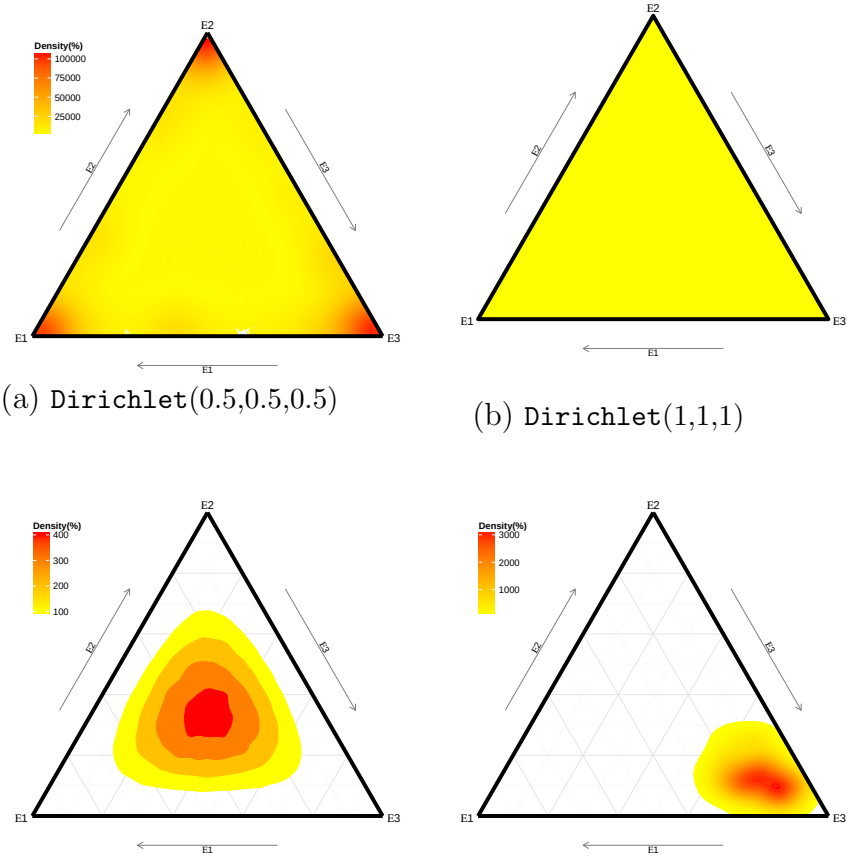
The notation $\boldsymbol{\varepsilon} \sim \texttt{Dirichlet}(\boldsymbol{\alpha})$ means that the K-dimensional variable $\boldsymbol{\varepsilon}$ is a random sample from the Dirichlet distribution with a K-dimensional vector of positive concentration parameters, $\boldsymbol{\alpha}$. The probability density function is,

$$\Pr(\boldsymbol{\varepsilon}|\boldsymbol{\alpha}) = \frac{\Gamma\big(\sum_{k=1}^{K} \alpha_k\big)}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \times \prod_{k=1}^{K} \varepsilon_k^{\alpha_k - 1}, \tag{A.1}$$

where $0 \leq \varepsilon_k \leq 1$ and $\varepsilon_1 + \varepsilon_2 + \ldots + \varepsilon_K = 1$. The mean, $\mathrm{E}(\varepsilon_k)$, and variance, $\mathrm{Var}(\varepsilon_k)$, of the kth element, ε_k, can be respectively obtained as follows

$$\mathrm{E}(\varepsilon_k) = \frac{\alpha_k}{\sum_{k=1}^{K} \alpha_k}, \qquad \mathrm{Var}(\varepsilon_k) = \frac{\alpha_k \sum_{a \neq k} \alpha_a}{\Big(\sum_{k=1}^{K} \alpha_k\Big)^2 \Big(\sum_{k=1}^{K} \alpha_k + 1\Big)}.$$

The Dirichlet distribution is usually skewed towards the element associated with the largest parameter component. Figure A.1 shows ternary plots illustrating the effect of different compositions of $\boldsymbol{\alpha}$ for a 3-dimensional Dirichlet example.



(a) `Dirichlet`(0.5,0.5,0.5)

(b) `Dirichlet`(1,1,1)

(c) `Dirichlet`(2,2,2)

(d) `Dirichlet`(3,3,15)

Figure A.1: Effect of various compositions of the $\boldsymbol{\alpha}$ parameter on the density of 3-dimensional variable $\boldsymbol{\varepsilon} = \{\varepsilon_1, \varepsilon_2, \varepsilon_3\}$ that jointly share a `Dirichlet`($\boldsymbol{\alpha}$) distribution. The plots were generated in `R` (R Core Team, 2013) with `gtools` (Warnes et al., 2014) and `ggtern`(Hamilton, 2014).

Appendix B

Code

This chapter contains the main HyPhy batch files and R scripts that were compiled during this dissertation. The attached pieces of code were written as functions. They are useful for inferring episodic directional selection (EDS) in protein coding sequences based on the algorithms described in the dissertation. The algorithms include collapsed Gibbs sampling (CGS), collapsed variational Bayes (CVB) and zero-order collapsed variational Bayes (CVB0). Some of the included files contain functions useful for performing EDS simulation studies.

The functions saved as `tFade` among the set of `R` and `HyPhy` scripts, embody majority of the other functions compiled for both platforms. The batch files have "`.bf`" extensions and contain more comprehensive code, while the `R` scripts have "`.R`" extensions. The `R` pieces of code were written to complement the results from `HyPhy`. Both versions are however useful. These scripts were written on `MacBook OS X 10.9.4 Mavericks`. For the batch files, `HyPhy` version `2.220140721beta(MPI)` for `Darwin on x86_64` was used while the `R` scripts were composed with `R` version `3.1.1` for `Platform: x86_64-apple-darwin13.1.0 (64-bit)`.

Section B.1 contains 17 batch files that include

`topology.bf`
`simulationFunction.bf`
`discreteSampling.bf`
`sorter.bf`
`favourFrequency.bf`
`initialGammas.bf`
`cvb0.bf`
`topicFade.bf`
`tFadeInterface.bf`.
`invariant.bf`
`mixedSimulation.bf`
`rateCategorization.bf`
`bayesFactor.bf`
`likelihoodCalculation.bf`
`cvb.bf`
`collapsedGibbs.bf`
`simAnalysisInterface.bf`

Section B.2 contains 2 R scripts including

`topicFade.R` `tFadeInterface.R`

Each of the included functions are accompanied by detailed instructions on how to use them. It is recommended to use the interface files. These include, `tFadeInterface.bf`, `tFadeInterface.R` and `simAnalysisInterface.bf`. The other ("engine") files may be consulted for more flexibility and more detailed function-specific instructions.

In order to avoid error messages when using the `HyPhy` functions, it is advisable to name variables with at least two characters e.g. use `aa = ??` instead of `a = ??`. It is also better to abstain from using fully capitalised variable names e.g. use `aA = ??` or `aa = ??` instead of `AA = ??`.

B.1 Hyphy code

`topology.bf` Simulate a tree topology

Description

The grand function in this file is called `genTopo` that simulates a balanced phylogeny.

Usage

```
genTopo(nLineages,branchLengthRange)
```

Arguments

`nLineages`: The number of the extant species. See **Details**.

`branchLengthRange`: A 2-dimensional vector that contains the range of the Uniform distribution where the branch lengths for the phylogeny should be randomly sampled.

Details

In order to ensure that the simulated phylogeny is balanced, `nLineages` must be a positive integer that may be expressed as 2^m, where m is also a positive integer. Otherwise, `nLineages` will be set to the maximum integer less than the supplied value that satisfies this requirement.

Value

A properly labelled tree topology in newick format.

Function

```
function giveBirth (n, branchLength){
  leafName = "Specie" + n + ":" + branchLength;
  return leafName;
}

function genTopo (nLineages, branchLengthRange){

  getBalance = Log(nLineages)/Log(2);
  nLevels = getBalance$1;
  validLineages = 2^nLevels;
  validLineages = Eval( Format(validLineages,0,0) );

  if(getBalance - nLevels > 0){
    warn = "" + nLineages + " taxa can not result in a balanced phylogeny" +
           " thus," + validLineages + " taxa were used instead!";
    fprintf(stdout, "\n\n", warn, "\n");
  }

  species = {};
  for(i=0; i<validLineages; i=i+1){
    bL = Random(branchLengthRange[0],branchLengthRange[1]);;
    species[i] = giveBirth(i+1, bL);
  }

  steps = nLevels;
  listCount = validLineages;
  nodeCount = 0;

  while(steps>0){
    listCount = listCount/2;
    newSpecies = {};

    for(i0=0; i0<listCount; i0=i0+1){
      bLength = Random(branchLengthRange[0],branchLengthRange[1]);
      newSpecies[i0] = "(" + species[i0*2] + "," + species[(i0*2)+1] +
                       ")iNode" + nodeCount + ":" + bLength;
      nodeCount = nodeCount + 1;
    }
    species = newSpecies;
    steps = steps - 1;
  }
  genTree = "" + "(" + species[0] + ")";
  return genTree;
}
```

Example

```
fprintf(stdout, genTopo(12, {{0.05,0.05}}), "\n\n");
fprintf(stdout, genTopo(8, {{0.05,0.09}}));
```

`invariant.bf` Get details of invariant sites

Description

This script contains a function called `invariantSites`. The function accepts a protein sequence of either amino acid or DNA residues. Let the number of sites in the input sequence be N. After successful implementation, the function returns an array of size N that summarises the composition of each site in the sequence.

Usage

```
invariantSites(sequenceBox,nState,oFolder)
```

Arguments

`sequenceBox:` a string that specifies the path to the location of a protein sequence alignment. See **Details**.

`nState:` a scalar that needs to be specified as four for a DNA sequence or twenty for an amino acid sequence.

`oFolder:` a string that gives the path to an existing folder where the output file must be saved. See **Details**.

Details

The sequence alignment in the supplied data path must use the one-letter IUPAC representation. If the user wishes to save the output file in the same folder where `invariant.bf` is saved, `oFolder` may simply be specified as an empty string i.e. `""`. However, if a subfolder, `y`, has been created for the outputs, `oFolder` should be entered as `"y/"`. The names of each element in the output array will correspond to zero-based indices of the associated site i.e. site one will be labelled `"0"`, site two will be labelled `"1"` and so on. The contents of the array will be protein residues and `"-"`. The residues will be attached to invariant sites and will represent the unique residue observed at the corresponding site, while `"-"` will appear next to non-invariant sites.

Value

An array of length N that will also be saved in `oFolder` as a text file named `invarianceInfo.txt`. See **Details**.

Function

```
function invariantSites (sequenceBox, nState, oFolder){
  aaId = {};
  if(nState==20){
    aa1iupac = "ACDEFGHIKLMNPQRSTVWY";
  }else{
    aa1iupac = "ACGT";
  }
  aa2iupac = "-" + aa1iupac;
  DataSet trialData = ReadDataFile(sequenceBox);

  count = 0;
  while(count < trialData.sites){
    commence = count;

    /* Break work to pieces - Maximum of 9 sites at a time */
    finir = {{count+8,trialData.sites-1}};
    finir = Min(finir,0);
    sitePack = "" + commence + "-" + finir + "";
    DataSetFilter trialFilter = CreateFilter(trialData, 1, sitePack);

    HarvestFrequencies(freq, trialFilter, trialFilter.sites, 1, 1);
    indicatorMat = freq["(_MATRIX_ELEMENT_VALUE_==1)*(_MATRIX_ELEMENT_ROW_+1)"];
    indicatorMat = ({1,nState}["1"]) * indicatorMat;
    ee = 0;
    for(dd=commence; dd<(finir+1); dd+=1){
      nId = indicatorMat[ee];
      aaId[dd] = aa2iupac[nId];
      ee = ee + 1;
    }
    count = finir + 1;
  }
  fprintf(""+oFolder+"invarianceInfo.txt", CLEAR_FILE, aaId);

  return 0;
}
```

Example

```
/* Amino acid example */
ligne1 = ">S1\nFSSHRTP\n>S2\nFSQERFP\n>S3\nFSAHRFP\n";
ligne2 = ">S4\nFSAHRAP\n>S5\nFSWHRFP\n>S6\nFSVHRFP";
```

```
dLignes = "" + ligne1 + ligne2;
DataSet pourVoire = ReadFromString(dLignes);
DataSetFilter pourVoireF = CreateFilter(pourVoire, 1);
fprintf("exampleData.txt", CLEAR_FILE, pourVoireF);
invariantSites("exampleData.txt", 20, "");
```

`simulationFunction.bf` Simulate alignment with all sites under EDS

Description

The function defined here is called `simulateEDsequence`. Its is useful for simulating protein sequence alignments that are subjected to EDS.

Usage

```
simulateEDsequence(nSites, nTaxa, tee, putBias, heteroRates,
                   baseRateMatrix, eFreqs, putBias0, oFolder)
```

Arguments

`nSites`: The number of sites that the simulated sequence should contain.

`nTaxa`: See `nLineages` under `topology.bf`.

`tee`: This needs to be specified as a vector that represents the range of a Uniform distribution (see `branchLengthRange` under `topology.bf`). It could also be entered as a path string. See **Details**.

`putBias`: A vector that describes how the incorporation of the branch-specific rate, $\mathcal{B}_n$, must be handled, where $n = 1, 2, \ldots,$ `nSites`. See **Details**.

`heteroRates`: A vector describing the site-specific rate, $\mathcal{A}_n$. See **Details**.

`baseRateMatrix`: A baseline substitution matrix. See **Details**.

`eFreqs`: A row vector that contains the equilibrium frequencies of the protein residues for the specified `baseRateMatrix`.

`putBias0`: A 2-dimensional vector that provides additional information on how to handle $\mathcal{B}_n$. See **Details**.

`oFolder`: See `invariant.bf`.

Details

If the element of `tee` is a string, the string must state the path to the newick file containing a phylogeny and its corresponding branch lengths. In such case, `nTaxa` becomes irrelevant and an unbalanced tree may be used. The only limitation to this approach is that the names of the foreground branches must be explicitly stated in `putBias`.

The first element of `putBias` could be a scalar specifying the proportion of the terminal branches that should be considered as the foreground branches. Its second element in such scenario must be, $\mathcal{B}_n$. The contents of `putBias` can also be specified as strings of the names of the nodes corresponding to the foreground branches. In such case, $\mathcal{B}_n$ must be specified as the last element in the `putBias` vector, else $\mathcal{B}_n$ will be considered as zero.

The dimension of the `heteroRates` vector could be one or two. If the dimension is one and the element is greater than zero, then $\mathcal{A}_n$ will be sampled independently for all n from a `Gamma` distribution whose scale and shape parameters will be the given scalar. If the dimension of the vector is two, the issued values will be respectively considered as the shape and scale parameters. When the first or only element of the vector is less than or equal to zero, $\mathcal{A}_n$ will be set equal to one.

The dimension of `baseRateMatrix` will indicate the type of data that needs to be simulated, where four will imply a DNA sequence while twenty will imply an amino acid sequence.

The first element of `putBias0` could be a non-negative scalar or a string prefixed by `"fixedTarget:"`. When a scalar is specified, it should state the cardinality of the set of target residues, $\mathbf{y}_n$. In such case, the elements of $\mathbf{y}_n$ will be randomly and independently sampled for each site. If the user prefers to fix the target residue(s) at all sites, the preferred target(s) need to be prefixed by `"fixedTarget:"` and supplied as the first element of `"putBias0"`. For example, `"fixedTarget:E"` will ensure that $\mathbf{y}_n = \{\mathtt{E}\}$, for all n. If the second element of `"putBias0"` is zero, then a phylogeny characterised by `less spread` background branches (as shown in Figure 4.20(a)) will be used. Otherwise, a `more spread` phylogeny will be used (see Figure 4.20(b)).

Value

`oFolder/siteMultiplier.txt`: $\mathcal{A}_n$ for all n.

`oFolder/simulatedData.txt`: The simulated protein sequence alignment saved in nexus format.

`oFolder/simulationTopology.txt`: The phylogeny used for the simulation.

`oFolder/simulatedAncestorData.txt`: The ancestral alignmment.

`oFolder/sitesTargets.txt`: $\mathbf{y}_n$ for all n.

`oFolder/biasedTaxa.txt`: The labels of the foreground branches.

`oFolder/invarianceInfo.txt`: See `invariant.bf`.

$\mathbf{y}_n$: The true set of target residues for all sites.

Function

```
/* Preamble */
/* <><><><> */
#include "topology.bf";
#include "invariant.bf";

function simulateEDsequence (nSites, nTaxa, tee, putBias, heteroRates,
                             baseRateMatrix, eFreqs, putBias0, oFolder)
{
  elapsed1C = Time(0);

  /* ---- Sort 'putBias0' ---- */
  clusteredForeground = putBias0[1];
  nFavours = putBias0[0];
  stringedFavours = "" + nFavours;
  stringOfFavours = stringedFavours[0][11];
  didNotFixTargetBases = stringOfFavours!="fixedTarget:";
  if(!didNotFixTargetBases){
    favourLength = Abs(stringedFavours);
    statedTargetBases = stringedFavours[12][favourLength];
    nFavours = Abs(statedTargetBases);
  }else{
    nFavours = nFavours * 1;
  }

  minBiasRate = 0;

  /* Identify size of residue states */
  /* ><><><><><><><><><><><><><><><><> */
  nStates = Columns(baseRateMatrix);

  if((nStates != 20) && (nStates != 4)){
    fprintf(stdout, "\nError: The baseline matrix supplied does not correspond",
```

```
          " to that of nucleotides (4 states) or amino acids (20 states)!\n");
}

/* Identify simulation characters */
/* ><><><><><><><><><><><><><><>< */
if(nStates == 20){
  rXters = {{"A", "C", "D", "E", "F", "G", "H", "I", "K", "L",
             "M", "N", "P", "Q", "R", "S", "T", "V", "W", "Y"}
            {"1","","","","","","","","","","","","","","","",
             "","","",""}};
  aa1iupac = "ACDEFGHIKLMNPQRSTVWY";
}else{
  rXters = {{"A", "C", "G", "T"}{"1", "", "", ""}};
  aa1iupac = "ACGT";
}

/* Construct tree topology */
/* <><><><><><><><><><><>< */
givenTopo = tee[0];
GetInformation(issuedRange, givenTopo);
issuedRange = Columns(issuedRange);
if(issuedRange){
  t1 = tee[0];
  t2 = tee[1];
  simTopology = genTopo(nTaxa, {{t1,t2}});
  Topology simTopo = simTopology;
  treeLengths = BranchLength(simTopo, -1);
  treeNames = BranchName(simTopo, -1);

  /* --- Tree must be balanced and bifurcating --- */
  usedTaxa = Log(nTaxa)/Log(2);
  usedTaxa = usedTaxa$1;
  usedTaxa = 2^usedTaxa;
  nTaxa = Eval( Format(usedTaxa,0,0) );
}else{
  fscanf(givenTopo, "Raw", givenTopo);
  simTopology = givenTopo;
  Topology simTopo = simTopology;
  treeLengths = BranchLength(simTopo, -1);
  treeNames = BranchName(simTopo, -1);
}

/* Simulate genetic sequence */
/* <><><><><><><><><><><><>< */
dataHive0 = "" + oFolder + "tempData.txt";
dataHive1 = "" + oFolder + "tempData.txt.anc";

/* ---- initialization ---- */
initRateMatrix = baseRateMatrix;
```

```
Model initWAG = (baseRateMatrix, eFreqs);
ACCEPT_ROOTED_TREES = 1;
Tree initTree = simTopo;
DataSet initSim = Simulate (initTree, eFreqs, rXters, 1, 1, dataHive0);

DataSet childrenData = ReadDataFile(dataHive0);
offspring = {};
for(child=0; child<childrenData.species; child=child+1){
  GetString(childName, childrenData, child);
  offspring[child] = "" + ">" + childName + "\n";
}

DataSet parentsData = ReadDataFile(dataHive1);
ancestors = {};
for(generation=0; generation<parentsData.species; generation=generation+1){
  GetString(parentName, parentsData, generation);
  ancestors[generation] = "" + ">" + parentName + "\n";
}
familySize = childrenData.species + parentsData.species;

/* ---- deal with bias incorporation ---- */
rateSpot = Columns(putBias) - 1;
biasRate = putBias[rateSpot];

GetInformation(issuedRate, biasRate);
issuedRate = Columns(issuedRate);
if(!issuedRate){
  biasRate = minBiasRate;
  fprintf(stdout, "\nWarning: No bias parameter was specified",
          " and ", biasRate, " has been used instead!\n");
}

fracType = putBias[0];
GetInformation(gaveFraction, fracType);
gaveFraction = Columns(gaveFraction);

/* ---- identify (random) foreground branches ---- */
if(gaveFraction){
  nBiased = putBias[0];
  nBiased = (nBiased * nTaxa) $ 1;
  startPoint = Random(0, nTaxa)$1;

  if(clusteredForeground){
    biasedTaxa = {1, nBiased}["_MATRIX_ELEMENT_COLUMN_"];
  }else{
    biasedTaxa = {1, nBiased}["_MATRIX_ELEMENT_COLUMN_*2"];
  }

  biasedTaxa = biasedTaxa + startPoint;
```

```
    biasedTaxa = biasedTaxa["(_MATRIX_ELEMENT_VALUE_)%(nTaxa)"];
    biasedTaxa = biasedTaxa
      ["_MATRIX_ELEMENT_VALUE_+((_MATRIX_ELEMENT_VALUE_==0)*nTaxa)"];

    biasedTaxons = {1, nBiased};
    for(ill=0; ill<nBiased; ill+=1){
      medication = biasedTaxa[ill];
      strong = "" + "Specie" + medication;
      biasedTaxons[ill] = strong;
    }
    biasedTaxa = biasedTaxons;
  }else{
    if(issuedRate){
      nBiased = Columns(putBias)-1;
    }else{
      nBiased = Columns(putBias);
    }
    biasedTaxa = {1, nBiased};
    for(taxaId=0; taxaId<nBiased; taxaId+=1){
      biasedTaxa[taxaId] = "" + putBias[taxaId];
    }
  }

  if(biasRate < 0){
    biasRate = minBiasRate;
    fprintf(stdout, "\nBias parameter is not allowed to be less than ",
            "0. ", biasRate, " has been used instead!\n");
  }

  if(gaveFraction && (!issuedRange)){
    biasRate = minBiasRate;
    biasedTaxa[0] = 0;
    fprintf(stdout, "\nTree topology was specified without giving the names",
            " of the foreground branches. Thus, no bias was incorporated!\n");
  }

  bbSizeFail = (nFavours>nStates) || (nFavours<0);
  if(bbSizeFail){
    nFavours = 1;
    fprintf(stdout, "\n\nWarning: The size of target residues per ",
            "site may not be less than zero or greater than the nu",
            "mber of states of the substitution matrix. One residu",
            "e has been targeted per site instead.\n");
  }

  /* ---- deal with rate multiplier incorporation ---- */
  LoadFunctionLibrary("ProbabilityDistributions");
  includeMultiplier = heteroRates[0];
  if(includeMultiplier < 0){
```

```
  fprintf(stdout, "\nThe parameter for Gamma distribution for the rate multiplier",
          " can not be < 0. As a result, rate multiplier was not incorporated in ",
          "the simulation!\n");
  includeMultiplier = 0;
}
caseOne = Columns(heteroRates);
caseTwo = Rows(heteroRates);
allCases = {{caseOne, caseTwo}};
givenCase = Max(allCases, 0);

if(includeMultiplier){
  alphaParameter = heteroRates[0];
  if(givenCase==2){
    betaParameter = heteroRates[1];
  }else{
    betaParameter = alphaParameter;
  }
  siteOmegas = {1, nSites}["sampleFromGamma(alphaParameter,betaParameter)"];
}else{
  siteOmegas = {1, nSites}["1"];
}

/* ---- main (site-by-site) simulation ---- */
randomTargets = {};
for(site=0; site<nSites; site=site+1){
  duration = Time(0);

  if(didNotFixTargetBases){

    // ---- randomly generate target residue(s) without repetition
    allBases = nStates;
    accruedTargets = "";
    stateIndex = {1,allBases}["_MATRIX_ELEMENT_COLUMN_"];
    for(bayzes=0; bayzes<nFavours; bayzes+=1){
      rIndex = Random(0,allBases)$1;
      newIndices = {1,allBases-1};
      meter = 0;
      for(lines=0; lines<allBases; lines+=1){
        if(rIndex!=lines){
          newIndices[meter] = stateIndex[lines];
          meter = meter + 1;
        }
      }
      newIndex = stateIndex[rIndex];
      stateIndex = newIndices;
      allBases = allBases - 1;
      rTarget = aa1iupac[newIndex];
      accruedTargets = accruedTargets + rTarget;
    }
```

```
}else{
  accruedTargets = statedTargetBases;
}
specialAA := "" + accruedTargets;
randomTargets[site] = "" + specialAA;

// ---- build substitution model
multiplier := siteOmegas__[site__];

newRateMatrix = {nStates,nStates};
simulMatrix = baseRateMatrix;
for(tag1=0; tag1<nStates; tag1=tag1+1){
  targetRow := specialAA$aa1iupac[tag1__];
  tRowYes := targetRow[0] + targetRow[1];

  for(tag2=0; tag2<nStates; tag2=tag2+1){
    targetCol := specialAA$aa1iupac[tag2__];
    tColYes := targetCol[0] + targetCol[1];

    if(tag1 != tag2){

      if(tRowYes<0 && tColYes>=0){
        newRateMatrix[tag1][tag2] := simulMatrix__[tag1__][tag2__] *
           ( (byas+(byas==0)) / ((1-(Exp(-1*byas))) + (byas==0)) ) *
           multiplier * mu;
      }
      if(tRowYes>=0 && tColYes<0){
        newRateMatrix[tag1][tag2] := simulMatrix__[tag1__][tag2__] *
              ( (byas + (byas==0)) / ((Exp(byas)-1) + (byas==0)) ) *
              multiplier * mu;
      }
      if((tRowYes<0 && tColYes<0) || (tRowYes>=0 && tColYes>=0)){
        newRateMatrix[tag1][tag2] := simulMatrix__[tag1__][tag2__] *
                                     multiplier * mu;
      }
    }
  }
}
Model WAG2001 = (newRateMatrix, eFreqs);

// ---- plant simulation tree
ACCEPT_ROOTED_TREES = 1;
Tree simTree = simTopo;

// ---- adjust simulation tree
allBranches = Columns(treeLengths) - 1;
for(branchI=0; branchI<allBranches; branchI=branchI+1){
  currName = treeNames[branchI];
  currLength = treeLengths[branchI];
```

```
    currConstraint = "simTree." + currName + ".mu := " + currLength;
    Eval(currConstraint);

    branchFlu = 0;
    for(vaccine=0; vaccine<nBiased; vaccine+=1){
      subject = biasedTaxa[vaccine];
      sick = currName == subject;
      branchFlu = branchFlu + sick;
    }

    if(branchFlu){
      restriction = "simTree." + currName + ".byas :=" + biasRate;
    }else{
      restriction = "simTree." +  currName + ".byas :=" + minBiasRate;
    }
    Eval (restriction);
  }

  // ---- simulate data
  DataSet familyData = Simulate (simTree, eFreqs, rXters, 1, 1);

  // ---- manoeuvre data for accurate reporting
  DataSetFilter familyFilter = CreateFilter(familyData, 1);
  GetInformation(familyString, familyFilter);

  for(child=0; child<childrenData.species; child=child+1){
    offspring[child] = offspring[child] + familyString[child];
  }

  generation1 = 0;
  for(lifeView=childrenData.species; lifeView<familySize; lifeView+=1){
    ancestors[generation1] = ancestors[generation1] + familyString[lifeView];
    generation1 = generation1 + 1;
  }
  duration1 = Time(0) - duration;
  situation = "" + "Simulated data for site " + (site+1) + " of " + nSites +
              " in " + Format(duration1,0,2) + " seconds.";
  fprintf(stdout, "\n", situation);
}

/* ><><>< Combine (site-by-site) data into complete sequence ><><>< */
offspringString = {};
ancestorsString = {};
if(nStates==20){
  offspringString[0] = "" + " $BASESET:BASE20\n";
  ancestorsString[0] = "" + " $BASESET:BASE20\n";
}else{
  offspringString[0] = "";
  ancestorsString[0] = "";
```

```
  }

  for(child=0; child<childrenData.species; child=child+1){
    offspringString[0] = offspringString[0] + "" + "\n" + offspring[child];
  }
  for(generation=0; generation<parentsData.species; generation=generation+1){
    ancestorsString[0] = ancestorsString[0] +"" +"\n" +ancestors[generation];
  }

  DataSet simulatedAA = ReadFromString(offspringString[0]);
  DataSetFilter simulatedAAfilter = CreateFilter(simulatedAA, 1);

  DataSet simulatedAncestor = ReadFromString(ancestorsString[0]);
  DataSetFilter simulatedAncestorFilter = CreateFilter(simulatedAncestor, 1);

  elapsed1Ca = Time(0) - elapsed1C;

  DATA_FILE_PRINT_FORMAT = 5;

  /* ><><>< Save results ><><>< */
  fprintf(oFolder+"biasedTaxa.txt", CLEAR_FILE, biasedTaxa);
  fprintf(oFolder+"siteMultiplier.txt", CLEAR_FILE, siteOmegas);
  fprintf(oFolder+"sitesTargets.txt", CLEAR_FILE, randomTargets);
  fprintf(oFolder+"simulatedData.txt", CLEAR_FILE, simulatedAAfilter);
  fprintf(oFolder+"simulationTopology.txt", CLEAR_FILE, "\n\n", simTopology);
  fprintf(oFolder+"simulatedAncestorData.txt",CLEAR_FILE,simulatedAncestorFilter);
  fprintf(stdout,"\n\nSimulation completed in ",Format(elapsed1Ca,0,2)," seconds\n");

  /* ><><>< Get details of invariant sites ><><>< */
  dataForest = oFolder + "simulatedData.txt";
  weird = invariantSites(dataForest, nStates, oFolder);

  return randomTargets;
}
```

Example

`Example 2` has been commented-out because it will cause the output files from `Example 1` to be over-written. To see results for `Example 2`, the user needs to comment-out `Example 1` and uncomment `Example 2`.

```
/* Example 1: Topology specification */
myTopo = "(((hello:0.1,hi:5),great:1),(nice:0.05,cape:1))";
fprintf("myTopo.txt", CLEAR_FILE, myTopo);

states = 04;
bMatrix = {states,states}["1"];
```

```
eqFreq = {1,states}["1/states"];
citeSize = 00005;
specieSize =  0008;
branchSize = {{"myTopo.txt"}};
biasVoice = {{"cape","hi", 000}};
heteroSites = {{0.5,0.5}};
simulateEDsequence(citeSize, specieSize, branchSize, biasVoice,
     heteroSites, bMatrix, eqFreq, {{"fixedTarget:CA",1}}, "");

// /* Example 2: Branch range specification */
// states = 20;
// bMatrix = {states,states}["1"];
// eqFreq = {1,states}["1/states"];
// citeSize = 00005;
// specieSize =  0018;
// branchSize = {{0.05,0.05}};
// biasVoice = {{0.25, 100}};
// heteroSites = {{0.5}};
// simulateEDsequence(citeSize, specieSize, branchSize, biasVoice,
//                    heteroSites, bMatrix, eqFreq, {{3, 0}}, "");
```

`mixedSimulation.bf` Simulate alignment with some sites under EDS

Description

This batch file contains the code for a function named `mixedBiasData`. The job of this function is to simulate a protein sequence alignment that has a proportion of its sites subjected to positive EDS. The remaining sites will be subjected to neutral EDS pressure.

Usage

```
mixedBiasData (mixtureVector, nSights, taxonSize, tea, biasMates,
               biasTaipe, gamaParams, baselineMath, eFreak, ansF)
```

Arguments

`mixtureVector`: A 2-dimensional vector. Its first element must state the proportion of the alignment sites that should be subjected to EDS, while the second element must give the corresponding $\mathcal{B}_n$ to be used.

`nSights`: See `nSites` in `simulationFunction.bf`.

`taxonSize`: See `nTaxa` in `simulationFunction.bf`.

`tea`: See `tee` in `simulationFunction.bf`.

`biasMates`: a vector that contains a scalar specifying the fraction of the terminal branches that should be considered as foreground branches. See **Details**.

`biasTaipe`: See `putBias0` in `simulationFunction.bf`.

`gamaParams`: See `heteroRates` in `simulationFunction.bf`.

`baselineMath`: See `baseRateMatrix` in `simulationFunction.bf`.

`eFreak`: See `eFreqs` in `simulationFunction.bf`.

`ansF`: See `oFolder` in `invariant.bf`.

Details

The contents of the `biasMates` vector may also be specified as the strings corresponding to the identifiers of the extant species attached to the terminal branches. The particular sites that are subjected to EDS are decided randomly.

Value

`ansF/siteBiases.txt`: A `nSights`-dimensional vector that contains $\mathcal{B}_n$ for all n.

See `simulationFunction.bf`.

Function

```
#include "simulationFunction.bf";

function mixedBiasData (mixtureVector, nSights, taxonSize, tea, biasMates,
                        biasTaipe, gamaParams, baselineMath, eFreak, ansF)
{

  nullDataFrac = mixtureVector[0];
  fracBug = (nullDataFrac>1) || (nullDataFrac<0);
  if(fracBug){
    fprintf(stdout, "\nError:\nThe proportion of sites to be generated with",
            " bias parameter of zero has been mis-specified. This value is ",
            "only allowed to be a non-negative value not greater than 1.\n");
  }

  parentDataHideOut = "" + ansF + "simulatedAncestorData.txt";
```

```
multiplierHideOut = "" + ansF + "siteMultiplier.txt";
varyInfoHideOut = "" + ansF + "invarianceInfo.txt";
babyDataHideOut = "" + ansF + "simulatedData.txt";
targetHideOut = "" + ansF + "sitesTargets.txt";
biasHideOut = "" + ansF + "siteBiases.txt";

if(!fracBug){
  nullDataSize = nullDataFrac * nSights;
  nullDataSize = nullDataSize$1;
  alternativeD = nSights - nullDataSize;

  mateSize = {{Columns(biasMates), Rows(biasMates)}};
  mateSize = Max(mateSize, 0);
  mateSizes = mateSize + 1;
  matesInfo = {1, mateSizes};
  for(bMates=0; bMates<mateSize; bMates+=1){
    matesInfo[bMates] = biasMates[bMates];
  }

  if(nullDataSize){
    fprintf(stdout, "\n=============================================\n",
                    "Simulating data with bias parameter value = 0.\n",
                    "=============================================");
    taggedResid0 = simulateEDsequence(nullDataSize, taxonSize, tea, matesInfo,
                                      gamaParams, baselineMath, eFreak,
                                      biasTaipe, ansF);

    DataSet nullHypoAncestor = ReadDataFile(parentDataHideOut);
    DataSet nullHypoData = ReadDataFile(babyDataHideOut);
    fscanf(multiplierHideOut, "Matrix", sightMultiply0);
    fscanf(varyInfoHideOut, "Raw", sightVary0);
    sightVary0 = Eval(sightVary0);
    siteBias0 = {1, nullDataSize};
    gbogboTarget = taggedResid0;
    gbogboBias = siteBias0;
  }

  if(alternativeD){
    fprintf(stdout, "\n=================================================\n",
                    "Simulating data with given positive bias parameter.\n",
                    "=================================================");
    positiveBias = mixtureVector[1];
    matesInfo[mateSize] = positiveBias;
    taggedResidB = simulateEDsequence(alternativeD, taxonSize, tea, matesInfo,
                                      gamaParams, baselineMath, eFreak,
                                      biasTaipe, ansF);

    DataSet haltHypoAncestor = ReadDataFile(parentDataHideOut);
    DataSet haltHypoData = ReadDataFile(babyDataHideOut);
```

```
  fscanf(multiplierHideOut, "Matrix", sightMultiplyB);
  siteBiasB = {1, alternativeD}["positiveBias"];
  fscanf(varyInfoHideOut, "Raw", sightVaryB);
  sightVaryB = Eval(sightVaryB);
  gbogboTarget = taggedResidB;
  gbogboBias = siteBiasB;
}

mixedSequence = nullDataSize * alternativeD;
if(mixedSequence){
  DataSet dataMixture = Concatenate(purge, nullHypoData, haltHypoData);
  DataSet originMixture = Concatenate(purge, nullHypoAncestor, haltHypoAncestor);

  gbogboMultiplier = {1, nSights};
  gbogboBias = {1, nSights};
  gbogboVaryInfo = {};
  gbogboTarget = {};

  for(onka=0; onka<nullDataSize; onka+=1){
    gbogboMultiplier[onka] = sightMultiply0[onka];
    gbogboTarget[onka] = taggedResid0[onka];
    gbogboVaryInfo[onka] = sightVary0[onka];
    gbogboBias[onka] = siteBias0[onka];
  }

  onka0 = 0;
  for(onka1=nullDataSize; onka1<nSights; onka1+=1){
    gbogboMultiplier[onka1] = sightMultiplyB[onka0];
    gbogboTarget[onka1] = taggedResidB[onka0];
    gbogboVaryInfo[onka1] = sightVaryB[onka0];
    gbogboBias[onka1] = siteBiasB[onka0];
    onka0 = onka0 + 1;
  }

  filterIndex = "";
  temporalB = gbogboBias;
  temporalT = gbogboTarget;
  temporalV = gbogboVaryInfo;
  temporalM = gbogboMultiplier;
  hindisses = {1, nSights}["_MATRIX_ELEMENT_COLUMN_"];
  permutedIndices = Random(hindisses,0);
  for(ounka=0; ounka<nSights; ounka+=1){
    vogueIndex = permutedIndices[ounka];
    filterIndex = "" + filterIndex + "," + vogueIndex;
    gbogboMultiplier[ounka] = temporalM[vogueIndex];
    gbogboVaryInfo[ounka] = temporalV[vogueIndex];
    gbogboTarget[ounka] = temporalT[vogueIndex];
    gbogboBias[ounka] = temporalB[vogueIndex];
  }
```

```
      fIndexLength = Abs(filterIndex);
      filterIndex = filterIndex[1][fIndexLength];
      DataSetFilter filteredKittens = CreateFilter(dataMixture, 1, filterIndex);
      DataSetFilter filteredBeavers = CreateFilter(originMixture, 1, filterIndex);

      fprintf(multiplierHideOut, CLEAR_FILE, gbogboMultiplier);
      fprintf(parentDataHideOut, CLEAR_FILE, filteredBeavers);
      fprintf(babyDataHideOut, CLEAR_FILE, filteredKittens);
      fprintf(varyInfoHideOut, CLEAR_FILE, gbogboVaryInfo);
      fprintf(targetHideOut, CLEAR_FILE, gbogboTarget);
      fprintf(biasHideOut, CLEAR_FILE, gbogboBias);
    }
  }
  return gbogboTarget;
}
```

Example

```
/* Example 1: Topology specification */
myTopo = "(((hello:0.1,hi:5),great:1),(nice:0.05,cape:1))";
fprintf("myTopo.txt", CLEAR_FILE, myTopo);

states = 04;
bMatrix = {states,states}["1"];
eqFreq = {1,states}["1/states"];
citeSize = 00005;
specieSize = 0008;
branchSize = {{"myTopo.txt"}};
biasVoice = {{"cape","hi"}};
heteroSites = {{0.5,0.5}};
mixVee = {{0.9, 100}};
tgs = mixedBiasData(mixVee, citeSize, specieSize, branchSize,
        biasVoice, {{1,1}}, heteroSites, bMatrix, eqFreq, "");
```

`discreteSampling.bf` Randomly sample from a discrete distribution

Description

The `sample0` function created in this file was designed to randomly sample from a discrete distribution.

Usage

```
sample0(observations,weights)
```

Arguments

`observations:` a vector of the values from where to sample.

`weights:` a vector of the weights associated with the elements of `observations`.

Value

The `sample0` function returns a random sample from the input vector, `observations`.

Function

```
function sample0 (observations, weights){

  if(Columns(weights)==1){
    weights = Transpose(weights);
  }
  if(Columns(observations)==1){
    observations = Transpose(observations);
  }

  weights = weights * (1/(+weights));

  cumWeights = weights["+weights[{{0,0}}][{{0,_MATRIX_ELEMENT_COLUMN_}}]"];
  u = Random(0,1);
  criterion = cumWeights["u/_MATRIX_ELEMENT_VALUE_"];
  criterion = criterion["_MATRIX_ELEMENT_VALUE_<=1"];
  chosen = observations["criterion[_MATRIX_ELEMENT_COLUMN_]==1"];
  sampled = (observations[chosen])[0];

  return sampled;
}
```

Example

```
values = {{1,2,3,4,5,6,7,8}};
nSample = 3000;
tested = {1, nSample}["sample0(values,values)"];

count = {1, Columns(values)};
tCount = {1, Columns(values)};
for(i=0; i<Columns(values); i+=1){
  currValue = values[i];
  identify = tested["_MATRIX_ELEMENT_VALUE_==currValue"];
  we = (+(identify))/nSample;
  count[i] = Format(we,0,4);
  tCount[i] = Format((values[i])*(1/(+values)),0,4);
}
fprintf(stdout, "\nSampled proportion:\n", count, "\nGiven weights:\n", tCount);
```

`rateCategorization.bf` Generate discrete rate categories

Description

The function defined here is called `splitRate`. It was designed to generate well-distributed parameter values that are useful for heterogeneous rate analyses. It was built as recommended by Murrell et al. (2013).

Usage

```
splitRate(nPoints, proportionLess1, upBound)
```

Arguments

`nPoints`: This must be entered as an integer and it must be greater than one. It corresponds to the number of parameter values preferred.

`proportionLess1`: This must be given as a proportion that represents the fraction of `nPoints` that should be set to values less than one.

`upBound1`: This argument should give the desired upper-bound for the parameter values.

Value

A column vector of parameter values will be returned after every successful implementation of the function.

Function

```
function splitRate(nPoints, proportionLess1, upBound){
  rateCats = {1, nPoints};
  nLess1 = (proportionLess1 * nPoints) $ 1;

  if(nLess1){
    rateCats = rateCats["_MATRIX_ELEMENT_COLUMN_/nLess1"];
  }
  rateCats = rateCats["_MATRIX_ELEMENT_VALUE_ * (_MATRIX_ELEMENT_COLUMN_<nLess1)"];
  rateCats = rateCats["_MATRIX_ELEMENT_VALUE_ + (_MATRIX_ELEMENT_COLUMN_==nLess1)"];

  pee = ((upBound-1)^(1/3)) / (nPoints-nLess1-1);
  tempVector = rateCats["(_MATRIX_ELEMENT_COLUMN_ - nLess1)"];
  tempVector = tempVector["(_MATRIX_ELEMENT_COLUMN_>nLess1)*(_MATRIX_ELEMENT_VALUE_)"];
  tempVector = tempVector["((_MATRIX_ELEMENT_VALUE_*pee)^3)"];
  tempVector = tempVector["_MATRIX_ELEMENT_VALUE_ + (_MATRIX_ELEMENT_COLUMN_>nLess1)"];
  rateCats = rateCats + tempVector;

  rateCats = Transpose(rateCats);
```

```
  return rateCats;
}
```

Example

```
tester = splitRate(10, 0.7, 50);
fprintf(stdout, tester);
```

`sorter.bf` Sort vector of numerical values in ascending order

Description

The function defined in this script is called `sort` and its duty is to arrange the numeric elements of a given vector in ascending order.

Usage

```
sort (givenVector)
```

Arguments

`givenVector`**:** The vector whose elements needs to be arranged.

Value

A vector that is similar to the supplied vector except that the contents of the output will be arranged in ascending order.

Function

```
function sort (givenVector) {

  transposed = 0;
  if(Columns(givenVector) == 1){
    transposed = 1;
    givenVector = Transpose(givenVector);
  }
  monitor = Columns(givenVector);

  excluder = Min(givenVector*(-1), 0);
  excluder = 2 * (-1) * excluder;

  sorted = {1, monitor};
  for(index=0; index<monitor; index+=1){
    smallest = Min(givenVector, 1);
```

```
    sorted[index] = smallest[0];
    takenSpot = smallest[1];
    givenVector[takenSpot] = excluder;
  }

  if(transposed == 1){
    sorted = Transpose(sorted);
  }

  return sorted;
}
```

Example

```
nSpecie = {{1}{90}{34}{3}{2}{87}{-09}{4}{-0.4}};
fprintf(stdout, sort(nSpecie));
```

`bayesFactor.bf` Calculate Bayes factors

Description

The function created here is named `bayesFactor`. It is useful for computing Bayes factors for a given posterior conditional probability matrix.

Usage

```
bayesFactor (postMat, nullGroup, appIndex, priorOddMat)
```

Arguments

`postMat`: A posterior conditional probability matrix whose column-specific Bayes factor is sought.

`nullGroup`: An integer vector that contains the indices of the rows of the supplied matrix that correspond to the null hypothesis. It must be zero-based. That is, the first row is identified by zero. For example, if the posterior probabilities attached to the alternative hypothesis are stored in rows $1, 3, 5, 8, 9$ and 10, in a 10-row matrix, then `nullGroup` must be entered as {{`1,3,5,6`}}.

`appIndex`: A scalar that indicates the approach to adopt when computing the prior odd. If zero is supplied, prior odd will be computed by dividing the number of rows that relates to the alternative hypothesis in the posterior probabilities matrix by the number of the rows related to the null hypothesis. If `appIndex`= 1, prior odd will be computed, using `postMat`, as the

sum of the probabilities in all rows attached to the alternative hypothesis across all columns, divided by the sum of the probabilities related to the null hypothesis. When `appIndex` equals two, it is necessary to input a prior probability matrix (as the fourth argument) that will be used to compute the prior odd using similar technique as described for when `appIndex`= 1.

`prioroddMat`: A matrix of identical dimension to `postMat`. It is only relevant if `appIndex` = 2. It will be interpreted as the posterior probability matrix of a sequence that is truly characterised by the null hypothesis and will be used to compute the prior odd.

Value

A vector of the computed column-specific Bayes factors.

Function

```
function bayesFactor (postMat, nullGroup, appIndex, priorOddMat){

  /* --- Standardize supplied probability matrix --- */
  colSums = ({1,Rows(postMat)}["1"]) * postMat;
  invColSums = colSums["1/_MATRIX_ELEMENT_VALUE_"];
  stdPostMat = postMat $ (({Rows(postMat),1}["1"]) * invColSums);

  if(Columns(nullGroup)==1){
    nullGroup = Transpose(nullGroup);
  }
  indicator = {Rows(stdPostMat),Columns(nullGroup)};
  indicator = indicator["_MATRIX_ELEMENT_ROW_==nullGroup[_MATRIX_ELEMENT_COLUMN_]"];
  indicator = indicator * ({Columns(indicator),1}["1"]);

  h0Post = Transpose(indicator) * stdPostMat;
  h0Post = h0Post["_MATRIX_ELEMENT_VALUE_+((_MATRIX_ELEMENT_VALUE_==0)*(1e-300))"];
  h1Post = (h0Post * (-1)) + 1;
  invH0Post = h0Post["1/_MATRIX_ELEMENT_VALUE_"];
  siteOdds = h1Post $ invH0Post;

  if(appIndex == 0){
    invPriorOdd = (+indicator) / (Rows(stdPostMat) - (+indicator));
  }
  if(appIndex == 1){
    invPriorOdd = (+h0Post) / (+h1Post);
  }
  if(appIndex == 2){
    h0Prior = Transpose(indicator) * priorOddMat;
    h0PriorSum = +(h0Prior);
    h0PriorSum = h0PriorSum + ((h0PriorSum==0)*(1e-300));
    oppIndicator = (indicator*(-1)) + 1;
    h1Prior = Transpose(oppIndicator) * priorOddMat;
```

```
    h1Prior = +(h1Prior);
    invPriorOdd = (+h0Prior) / (+h1Prior);
  }
  if((appIndex != 0) && (appIndex != 1) && (appIndex != 2)){
    bayesRatio = "\nError: The specified approach is invalid be" +
                 "cause, the fourth input is none of 0, 1, or 2.";
  }else{
    bayesRatio = siteOdds * invPriorOdd;
  }

  return bayesRatio;
}
```

Example

```
h0group = {{0,1}};
afterMat = {{0.230,0.000,0.150,0.170}
            {0.191,0.000,0.220,0.170}
            {0.193,0.000,0.210,0.240}
            {0.193,0.089,0.260,0.245}
            {0.193,0.911,0.160,0.175}};
h0mat = {Rows(afterMat),Columns(afterMat)}["1"];

tested = bayesFactor(afterMat, h0group, 0, "");
fprintf(stdout, "\nResult using approach 0:\n", tested);

tested1 = bayesFactor(afterMat, h0group, 1, "");
fprintf(stdout, "\nResult using approach 1:\n", tested1);

tested2 = bayesFactor(afterMat, h0group, 2, h0mat);
fprintf(stdout, "\nResult using approach 2:\n", tested2);
```

`favourFrequency.bf` Accelerated substitutions along foreground branches

Description

The function defined in this file is called `countFavours`. The function can be used to obtain the number of visible subsitutions towards the target residues along the foreground branches for all sites. It is particulaly useful for EDS analysis.

Usage

```
countFavours (extantData, extinctData, familyTree,
              blessedTaxa, theTargets, outFolder)
```

Arguments

`extantData`: a string specifying the path to the location of the file containing the extant sequences.

`extinctData`: a string that specifies the path to the location of the file that contains the ancestral sequences.

`familyTree`: a string that states the path to the location of the tree topology that gives the evolutionary relationship between the extant and ancestral sequences. The tree must be in newick format.

`blessedTaxa`: a vector of strings corresponding to the names of the terminal foreground branches.

`theTargets`: an array of the site-specific target residues. The labels of the element of the array must be the 0-based indices of the corresponding sites.

`outFolder`: see `oFolder` in `invariant.bf`

Value

`outFolder/table5plus.txt`: number of visible substitution towards the target residues along the foreground branches. See **Details**.

Details

The contents of the output file, `outFolder/table5plus.txt`, depends on the maximum cardinality of $\mathbf{y}_n$ over n. For example, a 3-site sequence whose site-specific targets are {"0":`A`,"1":`CD`,"2":`E`} will return two sets of counts corresponding to the following targets {{"`A`","`C`","`E`"}} and {{"","`D`",""}}.

Function

```
function countFavours (extantData, extinctData, familyTree,
                       blessedTaxa, theTargets, outFolder)
{
  /* ><>< Read sequences from specified locations  ><>< */
  DataSet oldSchoolData = ReadDataFile(extinctData);
  DataSetFilter oldSchoolFilter = CreateFilter(oldSchoolData, 1);
  GetString(oldSchoolNames, oldSchoolFilter, -1);

  DataSet newSchoolData = ReadDataFile(extantData);
  DataSetFilter newSchoolFilter = CreateFilter(newSchoolData, 1);
  GetString(newSchoolNames, newSchoolFilter, -1);

  taileFamille = newSchoolFilter.species + oldSchoolFilter.species;

  /* ><>< Get post-order arrangement ><>< */
```

```
fscanf(familyTree, "Raw", familyBond);
Topology familyTopology = familyBond;
postOrder = BranchName(familyTopology, -1);
numericPostOrder = Abs(familyTopology);

/* ><>< Concatenate data in post-order traversal ><>< */
GetInformation(newSchoolString, newSchoolFilter);
GetInformation(oldSchoolString, oldSchoolFilter);
allGenerations = {1, taileFamille};

for(aging=0; aging<oldSchoolFilter.species; aging+=1){
  dob = 0;
  stillSearching = 1;
  lostOne = oldSchoolNames[aging];
  while(stillSearching){
    lostsPics = postOrder[dob];
    hopeLost = lostOne % lostsPics;
    dob = dob + 1;
    stillSearching = !hopeLost;
  }
  throneQueue = dob - 1;
  allGenerations[throneQueue] = oldSchoolString[aging];
}

for(young=0; young<newSchoolFilter.species; young+=1){
  dob = 0;
  stillSearching = 1;
  lostOne = newSchoolNames[young];
  while(stillSearching){
    lostsPics = postOrder[dob];
    hopeLost = lostOne % lostsPics;
    dob = dob + 1;
    stillSearching = !hopeLost;
  }
  throneQueue = dob - 1;
  allGenerations[throneQueue] = newSchoolString[young];
}

/* ><>< Identify post-order location of the biased taxa ><>< */
myRuns = 0;
doNotStop = 1;
biasedFound = 0;
theBug = taileFamille * 2;
biasedSpots = blessedTaxa["theBug"];
while(doNotStop){
  lostBiased = blessedTaxa[biasedFound];

  for(theKids=0; theKids<taileFamille; theKids+=1){
    chosenKid = postOrder[theKids];
```

```
    foundThisBiased = lostBiased%chosenKid;

    if(foundThisBiased){
      biasedSpots[biasedFound] = theKids;
      biasedFound = biasedFound + 1;
      if(biasedFound == Columns(blessedTaxa)){
        theKids = taileFamille;
      }
    }
  }
  myRuns+ = 1;
  foundAllBiased = Max(biasedSpots,0) != theBug;
  doNotStop = (myRuns<Columns(blessedTaxa)) * (!foundAllBiased);
}
bugger = Max(biasedSpots,0) == theBug;
if(bugger){
  fprintf(stdout, "\nError: At least one of the labels of the biased ",
          "taxa does not match any of the taxa labels on the supplied",
          " topology. Please, adjust and re-run the function.");
}else{
  outName = ""+outFolder+"table5plus.txt";
  fprintf(outName, CLEAR_FILE, "\n");

  /* ><>< Account for multiple site-specific target residues ><>< */
  targetVector = {1,newSchoolFilter.sites};
  for(lesSaits=0; lesSaits<Columns(targetVector); lesSaits+=1){
    tempResids = theTargets[lesSaits];
    targetVector[lesSaits] = Abs(tempResids);
  }
  oscilation = Max(targetVector, 0);

  for(tsets=0; tsets<oscilation; tsets+=1){
    for(theseSite=0; theseSite<newSchoolFilter.sites; theseSite+=1){
      tempS3ng = theTargets[theseSite];
      targetVector[theseSite] = tempS3ng[tsets];
    }
    /* ><>< Compare targets to each of the concatenated sequences ><>< */
    targetBinary = {taileFamille, newSchoolFilter.sites};
    for(squeens=0; squeens<taileFamille; squeens+=1){
      currSequence = allGenerations[squeens];

      for(sigh=0; sigh<oldSchoolFilter.sites; sigh+=1){
        pair0 = targetVector[sigh];
        pair1 = currSequence[sigh];
        targetBinary[squeens][sigh] = pair0 == pair1;
      }
    }
    mraIndicator = targetBinary["!_MATRIX_ELEMENT_VALUE_"];
```

```
      /* ><>< Count number of visible substitutions along biased taxa ><>< */
      biasedMatrix = {Columns(blessedTaxa),newSchoolFilter.sites};
      for(lesBiased=0; lesBiased<Columns(blessedTaxa); lesBiased+=1){
        monBiasedIndex = biasedSpots[lesBiased];
        mostRecentAncestor = numericPostOrder[monBiasedIndex];
        monBiasedVect = targetBinary[monBiasedIndex][-1];
        mraVector = mraIndicator[mostRecentAncestor][-1];
        subsIdentifier = monBiasedVect + mraVector;
        for(arranger=0; arranger<newSchoolFilter.sites; arranger+=1){
          tempIndicator = subsIdentifier[arranger] == 2;
          biasedMatrix[lesBiased][arranger] = tempIndicator;
        }
      }

      /* ><>< Count site-specific visible subs. along all biased taxa ><>< */
      minCount = {1,newSchoolFilter.sites};
      for(kount=0; kount<newSchoolFilter.sites; kount+=1){
        minCount[kount] = +(biasedMatrix[-1][kount]);
      }
      fprintf(outName, "Site-specific target set (", tsets+1, "):", targetVector,
              "Site-specific minimum number of substitutions towards target set",
              " (", tsets+1, ") along foreground branches:", minCount, "\n\n");
    }
  }
  return 0;
}
```

Example

```
/* Extant sequences */
line1 = " $BASESET:BASE20\n>sp1\nTGWPM\n>sp2\nTGWPM\n>sp3\nTGMAM\n>sp4";
line2 = "\nTWKAM\n>sp5\nTMWPM\n>sp6\nTHWPM\n>sp7\nTGWPM\n>sp8\nTGWPM";
dataLines = "" + line1 + line2;
DataSet testExtSeq = ReadFromString(dataLines);
DataSetFilter testExtSeqF = CreateFilter(testExtSeq, 1);
fprintf("testExtSeq.txt", CLEAR_FILE, testExtSeqF);

/* Extinct sequences */
line3 = " $BASESET:BASE20\n>iN6\nTGWPM\n>iN4\nTGWPM\n>iN0\nTGWPM";
line4 = "\n>iN1\nTGMAM\n>iN5\nTGWPM\n>iN2\nTHWPM\n>iN3\nTGWPM";
datumLines = "" + line3 + line4;
DataSet testAncSeq = ReadFromString(datumLines);
DataSetFilter testAncSeqF = CreateFilter(testAncSeq, 1);
fprintf("testAncSeq.txt", CLEAR_FILE, testAncSeqF);

/* Tree topology */
fmlyT3 = "((((sp1,sp2)iN0,(sp3,sp4)iN1)iN4,((sp5,sp6)iN2,(sp7,sp8)iN3)iN5)iN6)";
fprintf("testTops.txt", CLEAR_FILE, fmlyT3);
```

```
/* Site-specific target residue */
fa4rd = {"0":"Y","1":"W","2":"K","3":"A","4":"V"};

/* Foreground taxa */
byeTaxa = {{"sp4","sp6"}};

/* Function implementation */
used = countFavours("testExtSeq.txt", "testAncSeq.txt", "testTops.txt",
                    byeTaxa, fa4rd, "");
```

`likelihoodCalculation.bf` Compute conditional likelihood

Description

This file contains a function defined as `preLikelyCompute`. The function was designed to estimate the conditional likelihood of any supplied alignment of protein sequences.

Usage

```
preLikelyCompute (dataBox, aaTargets, godOfBias, godOfOmega,
                  bPrior, xPrior, topoBox, bsTita, baseSmatrix,
                  oFolder, print)
```

Arguments

`dataBox`: A string that specifies the path to the location of a protein sequence alignment.

`aaTargets`: A string input that describes how target residues should accomodated during likelihood calculation. It may be specified as any one of `"Explore"`, `"Identify"`, or `"Fixed:??"`, where `??` is contained in {`A`,`C`,`G`,`T`} for a DNA sequence, or contained in {`A`,`C`,`D`,`E`,`F`,`G`,`H`,`I`,`K`,`L`,`M`, `N`,`P`,`Q`,`R`,`S`,`T`,`V`,`W`,`Y`} for an amino acid sequence. `"Explore"` will ensure that all the possible unambiguous IUPAC residues are analysed for each site in the given alignment. `"Identify"` will cause the given sequence to be split into sub-sequences. Sub-sequence `x` will comprise of sites that contain at least one `x` residue and will consequently be analysed with respect to target `x`. `"Fixed:??"` will ensure that only target(s) `"??"` is(are) considered in the conditional likelihood computation.

`godOfBias`: A vector that contains the inputs required to generate the branch rate categories as recommended by Murrell et al. (2013). It needs to be

of the form: {{`class size,fraction of points that should be less than one, upper bound`}}. `class size` should be at least two, otherwise two will be used.

`godOfOmega`: This vector is to site rate categories what `godOfBias` is to branch rates except that if its first element is less than two, site rates will be assumed to be homogeneous across all sites.

`bPrior`: A column vector of prior probabilities for the branch rate categories. Equal prior probabilities may be usedby simply specifying the first or only element of the vector of as `"Equal"`.

`xPrior`: A column vector of prior probabilities for the site rate categories. Its definition is similar to that of `bPrior`.

`topoBox`: This provides information on how to handle the topology associated with the supplied sequence. It could be specified in any of two forms. The first method only applies if one needs the branch lengths on the tree topology to be optimised. In such case one may simply specify a string stating the path to the location of the topology. Another permissible approach to do the same thing is to supply a vector of two elements, where the first element gives the path-to-topology string and the second element is zero. If `topoBox` is a vector and its second element is non-zero, the topology located in the supplied path will be assumed to include optimised branch lengths. It must be noted that the sequence and topology files need to be saved separately. That is, `DATAFILE_TREE` is not a valid path string.

`bsTita`: A vector of strings corresponding to the names of the foreground branches as they appear on the supplied topology and sequence alignment.

`baseSmatrix`: A matrix of baseline instantaneous substitution matrix. The dimension of this matrix will be used to determine the type of sequence supplied. A 4×4 implies that the given sequence alignment should be treated as DNA sequence, while a 20×20 implies that the alignment comprises of amino acids.

`oFolder`: A string that states the path to an existing folder where output files should be saved. If this input is supplied as `""`, all output files will be saved in the same folder where this batch file is saved. If it is desired to save all output files in a folder named `y` that is in the same location as this batch script, then this input should be entered as `"y/"`.

`print`: This input argument should one of `"True"` or `"False"`. When specified as `"True"`, detailed outputs will be saved as named text files in `oFolder`.

Value

After effective use of the function, the direct output will be an array of the computed conditional likelihood matrix/matrices. The key of the array will be the target residue(s) used for the respective matrix. Each matrix in the array will be of dimension (number of rate categories) × (number of [sub-]sequence sites). If site rate was accounted for during computation, the matrix rows will correspond to combination of branch and site rates that must be interpreted as follows:

		Site rate, $\mathcal{A}$			
		$\mathcal{A}_1$	$\mathcal{A}_2$		$\mathcal{A}_m$
Branch rate, $\mathcal{B}$	$\mathcal{B}_1$	row(1)	row(k+1)	...	row(mk-k+1)
	$\mathcal{B}_2$	row(2)	row(k+2)	...	row(mk-k+2)
	⋮	⋮	⋮	⋮	⋮
	$\mathcal{B}_k$	row(k)	row(k+k)	...	row(mk-k+k)

Two text files will be saved in `oFolder`. These include, `likelihoodKey.txt`: a file that contains an array with similar key to the returned likelihood matrix/matrices. The contents of the array will be zero-based indices of the sites analysed with respect to the corresponding target. The second output file is `categoriesUsed.txt`: a file that will contain values attached to the branch and site categories used for the likelihood computation. When `print` is `"True"`, an additional self-explanatory file named `allLikelyInfo.txt` will be saved in `oFolder`. It will contain details of the returned likelihood array.

Function

```
#include "rateCategorization.bf";

function preLikelyCompute (dataBox, aaTargets, godOfBias, godOfOmega, bPrior,
                           xPrior, topoBox, bsTita, baseSmatrix, oFolder, print)
{

  /* Preamble */
  /* ><><><>< */
  elapsed1E = Time(0);

  nCountry = Columns(baseSmatrix);
  if(nCountry == 20){
    aa1iupac = "ACDEFGHIKLMNPQRSTVWY";
  }else{
    aa1iupac = "ACGT";
  }
  empiricalRatesMatrix = baseSmatrix;
  DataSet analysisData = ReadDataFile(dataBox);
  trueSites = analysisData.sites;
```

```
/* Obtain estimated equilibrium frequencies */
/* ><><><><><><><><><><><><><><><><><><><><>< */
DataSetFilter analysisFilter = CreateFilter(analysisData, 1);
HarvestFrequencies(freqEsts, analysisFilter, 1, 1, 1);

estimatedFreqs = {};
for(freq=0; freq<Rows(freqEsts); freq=freq+1){
  aaCount = aa1iupac[freq];
  estimatedFreqs[aaCount] = freqEsts[freq];
}

/* Obtain optimized branch lengths */
/* <><><><><><><><><><><><><><><><><>< */
topoType = topoBox[1];
GetInformation(takenTopo, topoType);
takenTopo = Columns(takenTopo);

if(takenTopo==0){ topoBox1 = topoBox; }else{ topoBox1 = topoBox[0]; }
fscanf(topoBox1, "Raw", theTopology);

noLengths = (takenTopo==0) || (topoType==0);

if(noLengths){
  optimRatesMatrix = {nCountry, nCountry};

  for(tag1=0; tag1<nCountry; tag1=tag1+1){
    for(tag2=0; tag2<nCountry; tag2=tag2+1){
        optimRatesMatrix[tag1][tag2] := empiricalRatesMatrix__[tag1__][tag2__] * mu;
    }
  }
  Model baselineWAG = (optimRatesMatrix, freqEsts);
  ACCEPT_ROOTED_TREES = 1;
  Tree unratedTree = theTopology;

  LIKELIHOOD_FUNCTION_OUTPUT = 2;
  LikelihoodFunction unratedLikelihood = (analysisFilter, unratedTree);
  Optimize (unratedLikelyOutput, unratedLikelihood);

  optimizedTree = Format(unratedTree, 1, 1);
  optTreeName = BranchName(unratedTree, -1);
  optTreeLength = unratedLikelyOutput;
  nLocalParams = optTreeLength[1][1];
}else{
  ACCEPT_ROOTED_TREES = 1;
  Topology unratedTree1 = theTopology;

  optimizedTree = Format(unratedTree1, 1, 1);
  optTreeName = BranchName(unratedTree1, -1);
  optTreeLength = BranchLength(unratedTree1, -1);
```

```
  familyHead = Columns(optTreeLength) - 1;
  rootName = optTreeName[familyHead];
  optTreeLength[familyHead] = BranchLength(unratedTree1, rootName);
  nLocalParams = Columns(optTreeName) - 1;
}

/* Specify bias and multiplier categories using included rateCategorization.bf */
/* <><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>< */
biasFaith = (godOfBias[0])$1;
if(biasFaith <= 1){
  biasFaith = 2;
  fprintf(stdout, "\n", "Warning: size of bias categories can not be < 2. ",
          "A bias class of size ", biasFaith, " has been used instead!");
}
biasKlas = splitRate(biasFaith, godOfBias[1], godOfBias[2]);

if(godOfOmega[0] >= 2){
  godOfOmega[0] = (godOfOmega[0])$1;
  omegaKlas = splitRate(godOfOmega[0], godOfOmega[1], godOfOmega[2]);
  avoidZero = (omegaKlas[0] + omegaKlas[1]) / 2;
  omegaKlas[0] = avoidZero;
}else{
  omegaKlas = {{1}};
}

/* Deal with prior probabilities of rate categories */
/* ><><><><><><><><><><><><><><><><><><><><><><><><><><><><>< */
priors = {};
if(bPrior[0]=="Equal"){
  beePrior = {Rows(biasKlas),1}["1/Rows(biasKlas)"];
  priors["bias"] = beePrior;
}else{
  priors["bias"] = bPrior;
  bSum = bPrior * Transpose(bPrior["1"]);
  beePrior = bPrior["_MATRIX_ELEMENT_VALUE_*(1/bSum[0])"];
}

if(godOfOmega[0]=="True"){
  if(xPrior[0]=="Equal"){
    exPrior = {Rows(omegaKlas),1}["1/Rows(omegaKlas)"];
    priors["multiplier"] = exPrior;
  }else{
    priors["multiplier"] = xPrior;
    xSum = xPrior * Transpose(xPrior["1"]);
    exPrior = xPrior["_MATRIX_ELEMENT_VALUE_*(1/xSum[0])"];
  }
}else{
  exPrior = {{1}};
}
```

```
/* Specify substitution matrix */
/* <><><><><><><><><><><><><>< */
nBias = Rows(biasKlas);
bMin = Min(biasKlas, 0);
bMax = Max(biasKlas, 0);

nMultiplier = Rows(omegaKlas);
xMin = Min(omegaKlas, 0);
xMax = Max(omegaKlas, 0);

cLikeMatrix = {};
laSituation = 0;
cLikeMap = {};

for(aminoA=0; aminoA<Abs(aa1iupac); aminoA+=1){

  if(aaTargets%"Identify"){

    theseRezid = aa1iupac[aminoA];
    theseFilter = "" + "/[" + theseRezid + "]/";
    DataSetFilter groupFilter = CreateFilter(analysisData, 1, theseFilter);
    cLikeMap[theseRezid] = groupFilter.site_map;
    aaFavour := aa1iupac__[aminoA__];

  }

  if(aaTargets%"Explore"){

    theseRezid = aa1iupac[aminoA];
    aaFavour := aa1iupac__[aminoA__];
    DataSetFilter groupFilter = CreateFilter(analysisData, 1);
    cLikeMap[theseRezid] = "Every site in the specified sequence";

  }

  myUser = aaTargets[0][5];
  if(myUser%"Fixed:"){

    aminoA = Abs(aa1iupac);
    rezidDim = Abs(aaTargets);
    theseRezid = aaTargets[6][rezidDim];
    aaFavour := theseRezid;
    cLikeMap["TargetUsedForAllSites"] = theseRezid;
    DataSetFilter groupFilter = CreateFilter(analysisData, 1);

  }

  category multiplier = (nMultiplier, exPrior, MEAN, , omegaKlas, xMin, xMax);
```

```
eRmatrix := baseSmatrix;
newRmatrix = {nCountry, nCountry};

for(tag1=0; tag1<nCountry; tag1=tag1+1){
  targetRow := (aaFavour__)$(aa1iupac__[tag1__]);
  tRowYes := targetRow__[0] + targetRow__[1];

  for(tag2=0; tag2<nCountry; tag2=tag2+1){
    targetCol := (aaFavour__)$(aa1iupac__[tag2__]);
    tColYes := targetCol__[0] + targetCol__[1];

    if(tag1 != tag2){

      if(tRowYes<0 && tColYes>=0){
        newRmatrix[tag1][tag2] := eRmatrix__[tag1__][tag2__] * multiplier *
              ( (byas+(byas==0)) / ((1-(Exp(-1*byas))) + (byas==0)) ) * mu;
      }
      if(tRowYes>=0 && tColYes<0){
        newRmatrix[tag1][tag2] := eRmatrix__[tag1__][tag2__] * multiplier *
                ( (byas + (byas==0)) / ((Exp(byas)-1) + (byas==0)) ) * mu;
      }
      if( (tRowYes<0 && tColYes<0) || (tRowYes>=0 && tColYes>=0) ){
        newRmatrix[tag1][tag2] := eRmatrix__[tag1__][tag2__] *
                                  multiplier * mu;
      }
    }
  }
}

/* ---- Build branch-site heterogeneous model ---- */
Model newWAG = (newRmatrix, freqEsts);
ACCEPT_ROOTED_TREES = 1;
Tree newTree = optimizedTree;

category buyHas = (nBias, beePrior, MEAN, , biasKlas, bMin, bMax);

if(Columns(bsTita)==1){
  bsTita = Transpose(bsTita);
}

for(kay=0; kay<nLocalParams; kay+=1){
  specieTag = optTreeName[kay];
  specieAge = optTreeLength[0][kay];

  biasBranch = 0;
  for(sickTaxa=0; sickTaxa<Columns(bsTita); sickTaxa+=1){

    patient = bsTita[sickTaxa];
```

```
      exposed = patient == specieTag;
      biasBranch = biasBranch + exposed;

    }

    if(biasBranch){
      restriction = "newTree." + specieTag + ".byas := buyHas";
    }else{
      restriction = "newTree." + specieTag + ".byas :=" + bMin;
    }
    Eval (restriction);

    restrictions = "newTree." + specieTag + ".mu := " + specieAge;
    Eval(restrictions);
  }

  /* ---- Compute conditional likelihood ---- */
  LikelihoodFunction3 newLikelihood = (groupFilter, newTree, freqEsts);
  ConstructCategoryMatrix(likelyValue, newLikelihood, COMPLETE);
  cLikeMatrix[theseRezid] = likelyValue;

  laSituation = laSituation + groupFilter.sites;
  fprintf(stdout, "\nComputed conditional likelihood matrix up to site ",
          laSituation," of a maximum of ", nCountry*analysisFilter.sites,
          " sites.");

}
fprintf(stdout, "\n");

/* Time spent */
/* <><><><><> */
elapsed1Ea = Time(0) - elapsed1E;

/* Define what to output where */
/* <><><><><><><><><><><><><>< */
if(print=="True"){
  fprintf(""+oFolder+"allLikelyInfo.txt", CLEAR_FILE,"\nAnalyzed an alignment of ",
          analysisData.species," sequences with ",trueSites," sites.");
  fprintf(""+oFolder+"allLikelyInfo.txt","\n\nEstimated amino acid frequencies:\n",
          estimatedFreqs);
  if(noLengths){
    fprintf(""+oFolder+"allLikelyInfo.txt", "\n\nOptimized tree assuming no rate ",
            "variation:\n",unratedLikelihood);
  }else{
    fprintf(""+oFolder+"allLikelyInfo.txt","\n\nDetails of supplied tree:\nTopolo",
          "gy = \n", theTopology, "\n\nBranch lengths = ", optTreeLength);
    unratedLikelihood = "";
  }
  LIKELIHOOD_FUNCTION_OUTPUT = 4;
```

```
    fprintf(""+oFolder+"allLikelyInfo.txt", unratedLikelihood,"\n\nPrior weights of",
            " each rate category:\n",priors,"\n\nConditional likelihood estimate:\n",
             cLikeMatrix,"\n\nThe sequence site & target residue (given as the key)",
             " for each matrix of the full conditional likelihood matrices array:\n",
             cLikeMap,"\n\nDuration of conditional likelihood computation (seconds)",
             ":\n", Format(elapsed1Ea,0,2));
  }
  fprintf(""+oFolder+"categoriesUsed.txt",CLEAR_FILE,"\nBias categories:\n", biasKlas,
          "\n\n\nMultiplier categories:\n", omegaKlas);
  fprintf(""+oFolder+"likelihoodKey.txt", CLEAR_FILE, cLikeMap);


  return cLikeMatrix;
}
```

Example

```
line1 = ">S1\nATTCT\n>S2\nTTACT\n>S3\nGTTCA\n>S4\nGTTCA\n";
line2 = ">S5\nTTACA\n>S6\nTTACC\n>S7\nATTGT\n>S8\nTTACA";
thisDataset = line1 + line2;
DataSet testSeq = ReadFromString(thisDataset);
DataSetFilter testSeqF = CreateFilter(testSeq, 1);
fprintf("testSeq.txt", CLEAR_FILE, testSeqF);

topos = "((((S1,S2)iN0,(S3,S4)iN1)iN4,((S5,S6)iN2,(S7,S8)iN3)iN5)iN6)";
fprintf("testTopo.txt", CLEAR_FILE, topos);

nState = 04;
omgSize = 04;
buySize = 04;
fgTaxa = {{"S1","S7"}};
byePrior = {{"Equal"}};
ratePrior = {{"Equal"}};
dataSafe = "testSeq.txt";
toupology = {{"testTopo.txt",0}};
badLineMat = {nState,nState}["1"];
biasThetas = {{buySize, 0.7, 50}};
omegaTheta = {{omgSize, 0.7, 50}};
tRes = "Explore";

tester = preLikelyCompute(dataSafe, tRes, biasThetas, omegaTheta,
                          byePrior, ratePrior, toupology, fgTaxa,
                          badLineMat, "", "True");
fprintf(stdout, "\n", tester);
```

`initialGammas.bf` Initialise posterior weights

Description

The function described here is named `initialGammas`. It was designed for initialising the collapsed variational Bayes, zero-order callopsed variational Bayes and collapsed Gibbs sampling algorithms. The function was created following recommendations in Murrell et al. (2013).

Usage

```
initialGammas(bitter)
```

Arguments

`bitter`: A matrix of conditional likelihoods whose rows correspond to rate categories and the columns represent the different sites.

Value

A transformed matrix of similar dimension to the supplied conditional likelihood matrix.

Function

```
function initialGammas(bitter){
  colSums = ({1, Rows(bitter)}["1"]) * bitter;
  rowSums = bitter * ({Columns(bitter), 1}["1"]);
  invColsums = colSums["1/_MATRIX_ELEMENT_VALUE_"];
  iGamma = rowSums * invColsums;
  iGamma = iGamma $ ({Rows(bitter),Columns(bitter)}["Random(0.8,1.2)"]);
  leastP = Min(iGamma,0);
  if(leastP==0){
    iGamma = iGamma + ({Rows(bitter),Columns(bitter)}["Random(0.5,1)"]);
  }

  /* normalize matrix */
  colSums2 = ({1, Rows(bitter)}["1"]) * iGamma;
  colSums2 = colSums2["1/_MATRIX_ELEMENT_VALUE_"];
  colSums2 = ({Rows(bitter),1}["1"]) * colSums2;
  iGamma = iGamma $ colSums2;

  return iGamma;
}
```

Example

```
myMat = {{2,4,23,8}{56,0,10,11}{67,30,5,2}{38,16,25,44}};
fprintf(stdout, initialGammas(myMat));
```

`cvb.bf` Collapsed variational Bayes algorithm

Description

The function described here is useful for implementing the collapsed variational Bayes (CVB) algorithm for inferring episodic directional selection in protein coding sequences.

Usage

```
cvbMethod (beeter, hepsilon, alfas, iGamma, oFolder, giveAll)
```

Arguments

`beeter`: A conditional likelihood matrix for which inference is sought. The dimension of the matrix must be (number of rate categories) $\times$ (number of sites).

`hepsilon`: a scalar that states a preferred error tolerance level. The algorithm only terminates if the maximum difference between the elements of the mth and the $(m-1)$th posterior probability matrix is less than `hepsilon` or when $m = 1,000$.

`alfas`: A vector that states the prior probabilities attached to the corresponding rate categories.

`iGamma`: A matrix of initial probabilities that is required for initialising the approximation procedure.

`oFolder`: A string that specifies the path to an existing folder where printed outputs should be saved. This input is only relevant when the next input (that is, `giveAll`) is set as `"True"`.

`giveAll`: A `"True"` or `"False"` string that states if detailed output files should be saved in `oFolder`.

Value

A posterior probability matrix will be returned after the algorithm terminates. If `giveAll` was set to `"True"`, two text files with `".txt"` extension will be saved in `oFolder`. These include, (a.) `hCvbMat.txt`: a file that contains the returned posterior probability matrix, the number of iterations explored and the computation time in seconds. (b.) `hCvbDiff.txt`: a vector of the same length as the number of sequence sites. It will contain values of the maximum difference between the reported mth output and the $(m-1)$th matrix for each column.

Function

```
function cvbMethod (beeter, hepsilon, alfas, iGamma, oFolder, giveAll){
  elapsed1J = Time(0);

  /* Prior distribution of data over rate classes */
  /* ><><><><><><><><><><><><><><><><><><><><><><>< */
  if(Rows(alfas)>1){
    alfas = Transpose(alfas);
  }

  /* Initialization */
  /* ><><><><><><>< */
  newGama = Transpose(iGamma);

  nIteration = 0;
  maxDiff = hepsilon + 1;
  usedBeta = Transpose(beeter);

  while(maxDiff > hepsilon && nIteration < 1000){
    oldGama = newGama;

    GamaColumnSums = ({1, Rows(newGama)}["1"]) * newGama;
    excludeGammaKj = (({Rows(newGama),1}["1"]) * GamaColumnSums) - newGama;
    leftBracket = (({Rows(newGama),1}["1"]) * alfas) + excludeGammaKj;

    leftSide = usedBeta$leftBracket;

    rBracesTop1 = newGama$(- newGama + 1);
    top1colsums = ({1, Rows(newGama)}["1"]) * rBracesTop1;
    top1lessKj = (({Rows(newGama),1}["1"]) * top1colsums) - rBracesTop1;

    rBracesBelow = (leftBracket$leftBracket) * 2;
    rBracesBelow = rBracesBelow["(-1)/(_MATRIX_ELEMENT_VALUE_)"];

    rightBraces = top1lessKj$rBracesBelow;
    rightSide = rightBraces["Exp(_MATRIX_ELEMENT_VALUE_)"];

    newGama = leftSide$rightSide;
```

```
    /* Standardize posterior probabilities */
    rowSums = newGama * ({Columns(newGama),1}["1"]);
    invRowSums = rowSums["1/_MATRIX_ELEMENT_VALUE_"];
    cvbGama = (invRowSums * ({1, Columns(newGama)}["1"]))$newGama;
    newGama = cvbGama;

    differ = {1, Rows(newGama)};

    diffMat = (oldGama - newGama)["Abs(_MATRIX_ELEMENT_VALUE_)"];
    for(ab=0; ab<Rows(newGama); ab+=1){
      vect = diffMat[ab][-1];
      differ[ab] = Max(vect, 0);
    }
    maxDiff = Max(differ, 0);

    nIteration = nIteration + 1;

    update = "Maximum difference after " + nIteration + " iterations = " + maxDiff;
    fprintf(stdout, update, "\n");
  }
  cvbGama = Transpose(newGama);

  /* Computation time */
  /* ><><><><><><><>< */
  elapsed1Ja = Time(0) - elapsed1J;

  /* Specify what and where to print */
  /* <><><><><><><><><><><><><><><>< */
  if(giveAll%"True"){

    fprintf(""+oFolder+"hCvbDiff.txt", CLEAR_FILE, differ);
    fprintf(""+oFolder+"hCvbMat.txt",CLEAR_FILE,"\n\nCVB posterior probabilities:\n",
            cvbGama, "\n\nNumber of iterations:\n", nIteration,"\n\nCVB completion ",
            "time (secs):\n", Format(elapsed1Ja, 0, 4));

  }

  return cvbGama;
}
```

Example

```
bettas = {{0.000038133,0.005280549,0.000252946,0.000004776,0.000095518}
          {0.000061862,0.004688458,0.000361064,0.000006283,0.000131391}
          {0.000094077,0.004074869,0.000487594,0.000007960,0.000171503}
          {0.000864087,0.000000001,0.001534776,0.000001926,0.000453832}};
```

```
Gama0 = {Rows(bettas),Columns(bettas)}["1"];
halpha = {1,Rows(bettas)}["0.5"];
hepsilon = 1e-20;
tested = cvbMethod(bettas, hepsilon, halpha, Gama0, "", "True");
fprintf(stdout, tested);
```

`cvb0.bf` Zero-order collapsed variational Bayes algorithm

Description

The function described here is useful for implementing the zero-order collapsed variational Bayes (CVB) algorithm for inferring episodic directional selection in protein coding sequences.

Usage

```
cvbZero (beeter, hepsilon, alfas, iGamma, oFolder, wantAll)
```

Arguments

`beeter:` See `cvb.bf` above.

`hepsilon:` See `cvb.bf` above.

`alfas:` See `cvb.bf` above.

`iGamma:` See `cvb.bf` above.

`oFolder:` See `cvb.bf` above.

`wantAll:` See `giveAll` in `cvb.bf` above.

Value

A zero-order analogue of the outputs described for `cvb.bf` above.

Function

```
function cvbZero (beeter, hepsilon, alfas, iGamma, oFolder, wantAll){
  elapsed1I = Time(0);

  /* Prior distribution of data over rate classes */
  /* ><><><><><><><><><><><><><><><><><><><><><><><><><>< */
  if(Rows(alfas)>1){
    alfas = Transpose(alfas);
  }
```

```
/* Initialization */
/* ><><><><><><>< */
newGama = Transpose(iGamma);
runningBeta = Transpose(beeter);
nIteration = 0;
maxDiff = hepsilon + 1;

while(maxDiff > hepsilon && nIteration < 1000){
  oldGama = newGama;

  GamaRowSums = ({1, Rows(newGama)}["1"]) * newGama;
  excludeGammaKj = (({Rows(newGama),1}["1"]) * GamaRowSums) - newGama;
  leftBracket = (({Rows(newGama),1}["1"]) * alfas) + excludeGammaKj;

  newGama = runningBeta$leftBracket;

  /* Standardize posterior probability */
  rowSums = newGama * ({Columns(newGama),1}["1"]);
  invRowSums = rowSums["1/_MATRIX_ELEMENT_VALUE_"];
  cvb0Gama = (invRowSums*({1,Columns(newGama)}["1"]))$newGama;
  newGama = cvb0Gama;

  differ = {1, Rows(newGama)};
  diffMat = (oldGama - newGama)["Abs(_MATRIX_ELEMENT_VALUE_)"];
  for(ab=0; ab<Rows(newGama); ab+=1){
    vect = diffMat[ab][-1];
    differ[ab] = Max(vect, 0);
  }
  maxDiff = Max(differ, 0);

  nIteration = nIteration + 1;

  update = "Maximum difference after " + nIteration + " iterations = " + maxDiff;
  fprintf(stdout, update, "\n");
}
cvb0Gama = Transpose(newGama);

/* Computation time */
/* ><><><><><><><>< */
elapsed1Ia = Time(0) - elapsed1I;

/* Specify what and where to print */
/* <><><><><><><><><><><><><><><>< */
if(wantAll%"True"){

  fprintf(""+oFolder+"hCvb0Diff.txt", CLEAR_FILE, differ);
  fprintf(""+oFolder+"hCvb0Mat.txt",CLEAR_FILE,"\n\nCVB0 posterior probabilities:\n",
          cvb0Gama, "\n\nNumber of iterations:\n", nIteration, "\n\nCVB0 completion",
          " time (secs):\n",  Format(elapsed1Ia, 0, 4));
```

```
  }

  return cvb0Gama;
}
```

Example

```
bettas = {{0.000038133,0.005280549,0.000252946,0.000004776,0.000095518}
          {0.000061862,0.004688458,0.000361064,0.000006283,0.000131391}
          {0.000094077,0.004074869,0.000487594,0.000007960,0.000171503}
          {0.000864087,0.000000001,0.001534776,0.000001926,0.000453832}};

Gama0 = {Rows(bettas),Columns(bettas)}["1"];
halpha = {1,Rows(bettas)}["0.5"];
hepsilon = 1e-20;
tested = cvbZero(bettas, hepsilon, halpha, Gama0, "", "True");
fprintf(stdout, tested);
```

`collapsedGibbs.bf` Collapsed Gibbs sampling algorithm

Description

The function defined here is designed for approximating the posterior probability distribution of individual sites in a sequence alignment over specific set of substitution rate categories, using the collapsed Gibbs sampling algorithm.

Usage

```
collapsedGibbs (finalSize, tSize, bSize, bttas, alfas, iGamma,
          H0group, bFmeth, pOddMat, saveAll, oFolder, printAll)
```

Arguments

`finalSize`: A scalar that indicates the total Gibbs posterior probability matrix samples desired after thinning and burning-in.

`tSize`: A scalar that represents the thin size.

`bSize`: A scalar that gives the burn-in size.

`bttas`: See `beeter` under `cvb.bf` above.

`alfas`: See `cvb.bf` above.

`iGamma`: See `cvb.bf` above.

`H0group`: A zero-based vector that gives the rows of `bttas` attached to the rate classes of the null hypothesis. See `nullGroup` under `bayesFactor` for more details.

`bFmeth`: A scalar that must be an element of $\{0, 1, 2\}$. It indicates the prior odd computation approach that must be used for computing the Bayes factors that will be returned. See `appIndex` under `bayesFactor` for more details.

`pOddMat`: See `priorOddMat` under `bayesFactor` for more details.

`saveAll`: A `"True"` or `"False"` string that states whether to save all the Gibbs draws.

`oFolder`: A string that gives the path to a folder where output files must be saved.

`printAll`: A `"True"` or `"False"` string that indicates whether output files should be saved.

Value

After successful implementation, a posterior conditional probability matrix will be returned. Whenever `printAll` is supplied as `"True"`, four text files will be saved in `oFolder`. These include,

(a.) `hGibbsPostP.txt`: a matrix of the Gibbs posterior conditional probabilities and the execution time in seconds.

(b.) `hGibbsPpSd.txt`: a matrix of the standard deviation estimated from the Gibbs sampled matrices, after the samples were thinned and burnt-in.

(c.) `hGibbsBfs.txt`: a vector of the Bayes factors for each column of the supplied likelihood matrix. Bayes factors will be estimated for every sampled posterior probability matrix. However, the recorded values will be the mean estimates after thinning and burning-in. If `saveAll` is `"True"`, this file will be saved in `oFolder` even if `printAll` is not `"True"`.

(d.) `hGibbsBfsSd.txt`: a vector containing the standard deviation of the factors whose mean was saved in `hGibbsBfs.txt`.

Function

```
/* Preamble */
/* ><><><>< */
#include "bayesFactor.bf";
#include "discreteSampling.bf";

function collapsedGibbs (finalSize, tSize, bSize, bttas, alfas, iGamma,
                         H0group, bFmeth, pOddMat, saveAll, oFolder, printAll)
{
```

```
elapsed1H = Time(0);

/* Initialization */
/* ><><><><><><>< */
if(saveAll=="True"){
  fprintf(""+oFolder+"hAllGibbsBFs.txt", CLEAR_FILE, "\n{");
}

indexCage = {1, finalSize}["_MATRIX_ELEMENT_COLUMN_*tSize"];
indexCage = indexCage + bSize;
totalDraw = indexCage[finalSize-1];
mainCount = 0;
mainTag = indexCage[mainCount];
avgGibbs = {Rows(iGamma), Columns(iGamma)};
stdDevGibbs = avgGibbs;
avgGibbsBFs = {1, Columns(iGamma)};
stdDevGibbsBFs = avgGibbsBFs;

/* Prior distribution of sequence over rate classes */
/* ><><><><><><><><><><><><><><><><><><><><><><><><><><><><>< */
if(Rows(alfas)>1){
  alfas = Transpose(alfas);
}

/* Initialization */
/* ><><><><><><>< */
newGama = iGamma;
drawn = 0;
classes = {1, Rows(bttas)}["_MATRIX_ELEMENT_COLUMN_"];

while(drawn < (totalDraw+1)){

  siteCats = {1, Columns(newGama)};
  siteCats = siteCats["sample0(classes,newGama[-1][_MATRIX_ELEMENT_COLUMN_])"];

  allOnes = {Columns(newGama),1}["1"];
  nId = newGama["_MATRIX_ELEMENT_ROW_==siteCats[_MATRIX_ELEMENT_COLUMN_]"];
  nVect = nId * allOnes;
  nLessImatrix = (nVect*Transpose(allOnes)) - nId;

  newGama = allOnes * alfas;
  newGama = Transpose(newGama) + nLessImatrix;
  newGama = newGama$bttas;

  /* --- Standardize sampled probability matrix --- */
  colSums = ({1,Rows(newGama)}["1"]) * newGama;
  invColSums = colSums["1/_MATRIX_ELEMENT_VALUE_"];
  newGama = newGama $ (({Rows(newGama),1}["1"]) * invColSums);
```

```
  /* --- Manipulate and store sampled probability matrix --- */
  gibbsSamples = newGama;
  newBFs = bayesFactor(gibbsSamples, H0group, bFmeth, pOddMat);

  if(saveAll=="True"){
    oldBeefs = "" + newBFs;
    theClock = Abs(oldBeefs);
    agingBeefs = oldBeefs[2][theClock-3];
    fprintf(""+oFolder+"hAllGibbsBFs.txt", agingBeefs);
  }

  if(mainTag == drawn){
    avgGibbs = avgGibbs + gibbsSamples;
    sqGibbs = gibbsSamples $ gibbsSamples;
    stdDevGibbs = sqGibbs + stdDevGibbs;

    avgGibbsBFs = avgGibbsBFs + newBFs;
    sqGibbsBFs = newBFs $ newBFs;
    stdDevGibbsBFs = sqGibbsBFs + stdDevGibbsBFs;

    mainCount = mainCount + 1;
    if(mainCount < finalSize){
      mainTag = indexCage[mainCount];
    }
  }
  drawn = drawn + 1;
  update = "\nSampled posterior probability matrix "+ drawn +" of "+(totalDraw+1);
  fprintf(stdout, update);
}

if(saveAll=="True"){
  fprintf(""+oFolder+"hAllGibbsBFs.txt", "}\n");
}

stdDevGibbs = (stdDevGibbs-((avgGibbs$avgGibbs)*(1/finalSize)))*(1/(finalSize-1));
stdDevGibbs = stdDevGibbs["Sqrt(_MATRIX_ELEMENT_VALUE_)"];
avgGibbs = avgGibbs * (1/finalSize);

stdDevGibbsBFs = (stdDevGibbsBFs - ((avgGibbsBFs$avgGibbsBFs) * (1/finalSize))) *
                 (1/(finalSize-1));
stdDevGibbsBFs = stdDevGibbsBFs["Sqrt(_MATRIX_ELEMENT_VALUE_)"];
avgGibbsBFs = avgGibbsBFs * (1/finalSize);


/* Computation time */
/* ><><><><><><><>< */
elapsed1Ha = Time(0) - elapsed1H;
elapsed1Hb = Eval( Format(elapsed1Ha,0,4) );
geeTime = "" + "Execution Time (Seconds): " + elapsed1Hb;
```

```
  /* Specify where to save/print which result */
  /* ><><><><><><><><><><><><><><><><><><><>< */
  if(printAll%"True"){

    fprintf(""+oFolder+"hGibbsPpSd.txt", CLEAR_FILE, "\n", stdDevGibbs);
    fprintf(""+oFolder+"hGibbsBfs.txt", CLEAR_FILE, "\n", avgGibbsBFs);
    fprintf(""+oFolder+"hGibbsBfsSd.txt", CLEAR_FILE, "\n", stdDevGibbsBFs);
    fprintf(""+oFolder+"hGibbsPostP.txt", CLEAR_FILE, "\n", avgGibbs, "\n", geeTime);

  }

  return avgGibbs;
}
```

Example

```
bettas = {{0.000038133,0.005280549,0.000252946,0.000004776,0.000095518}
          {0.000061862,0.004688458,0.000361064,0.000006283,0.000131391}
          {0.000094077,0.004074869,0.000487594,0.000007960,0.000171503}
          {0.000864087,0.000000001,0.001534776,0.000001926,0.000453832}};

Gama0 = {Rows(bettas),Columns(bettas)}["1"];
hallfars = {1,Rows(bettas)}["0.5"];
nullClass = {{0}};
deez = collapsedGibbs(100, 1, 0, bettas, hallfars, Gama0,
                      nullClass, 0, "", "True", "", "True");
fprintf(stdout, "\n", deez);
```

`topicFade.bf` Topic model-based approach to inferring EDS

Description

The function, `tFade`, that is described here is a comprehensive function that may be used to infer episodic directional selection in protein sequences using the topic model-based algorithms that were developed in this dissertation.

Usage

```
tFADE (seqSafe, nBiasCat, nXcat, fgTaxa, h0clas, luckyAa,
       topoSafe, meth, bfMeth, instantRmatrix, anStore, p2param)
```

Arguments

`seqSafe`: A string that specifies the path to the location of the sequence that needs to be analysed.

`nBiasCat`: A scalar that states the preferred number of branch rate categories.

`nXcat`: A scalar that represents the preferred number of site rate categories.

`fgTaxa`: A vector that contains the labels of all the foreground branches. The vector contents must be strings that each corresponds to the name of a foreground branch as it appeared in the given sequence alignment.

`h0clas`: a vector that describes the branch rate categories associated with the null hypothesis. Its first element may be given as `"Exact"` or `"CutOff"`. If `"Exact"` is used, the vector is allowed to be of any length. All the numeric values from the second to the last element of the given vector will then be interpreted as actual branch rate values. When `"CutOff"` is specified, the vector is expected to contain two elements. In such scenario, the second element is considered as a branch rate threshold such that the null hypothesis is described by every category associated with a value less than the supplied value.

`luckyAa`: A string that could be one of `"Identify"`, `"Explore"` or, `"Fixed:??"`, where `??` needs to be contained in {`A`,`C`,`D`,`E`,`F`,`G`,`H`,`I`,`K`,`L`,`M`,`N`,`P`,`Q`,`R`,`S`,`T`,`V`,`W`,`Y`} for an amino acid sequence or contained in {`A`,`C`,`G`,`T`} for a nucleotide sequence. See `aaTargets` in `likelihoodCalculation.bf` for further details.

`topoSafe`: A string input that characterises the tree topology. It could be simply specified as a string that identifies the location of the file containing the topology. It could also be specified as a vector of length two, where the first element is a string that gives the path to the tree topology file, and the other element is a `0` or `1` indicator of whether the topology contains optimised branch lengths.

`meth`: An indicator vector that contains three `0` or `1` elements. The first element indicates (when given as `1`) whether to apply the collapsed Gibbs sampling algorithm. Similarly, the other two elements of the vector correspond to collapsed variational Bayes and zero-order collapsed variational Bayes algorithms respectively.

`bfMeth`: A scalar that must be an element of $\{0, 1, 2\}$. It indicates the prior odd computation approach that must be used for the Bayes factors that will be returned. See `appIndex` under `bayesFactor.bf` for more details.

`instantRmatrix`: A baseline instantaneous substitution matrix. The type of sequence residues (that is, amino acid or nucleotide) will be deduced from the dimension of the given matrix.

`anStore`: A string that states the path to a folder where output files may be saved.

`p2param`: An array that is meaningful only when `bfMeth=2`. Its key is case sensitive and must be with respect to each of the initiated approaches. If all the three approaches are initiated, the keys must be `"Gibbs"`, `"CVB"` and `"CVB0"`. Each key needs to hold the matrix of prior probabilities required to compute the prior odd for its respective approach.

`Note`: For further details on the implications of the fixed parameters in the body of the function, please see the notes written in the included batch files.

Value

After successful implementation, distinctly named output files will be saved in `anStore`. These output files will include initial probability matrices, site-specific Bayes factors and posterior probability matrices. If collapsed Gibbs sampling algorithm was activated, the reported estimates will be average of the Gibbs draws after burning-in and thinning. Zero-based indices of rows of the posterior probability matrix that were considered to by governed by the null hypothesis, during computation of Bayes factors, will be saved as in `nullClass.txt`.

Function

```
/* ><>< 	Include external batch files 	><>< */
#include "likelihoodCalculation.bf";
#include "rateCategorization.bf";
#include "collapsedGibbs.bf";
#include "initialGammas.bf";
#include "bayesFactor.bf";
#include "invariant.bf";
#include "sorter.bf";
#include "cvb0.bf";
#include "cvb.bf";

function tFADE (seqSafe, nBiasCat, nXcat, fgTaxa, h0clas, luckyAa, topoSafe,
                meth, bfMeth, instantRmatrix, anStore, p2param)
{
  smallestBiasValue = 0;
  bugged = 0;

  /* ><><>< Null group definition ><><>< */
  biasMaker = {{nBiasCat, 0.7, 0050}};          /* Bias categorization parameters */
  xMaker = {{nXcat, 0.7, 0050}};                 /* Multiplier classes parameters */
  mxProb = {{"Equal"}};                          /* Prior probability: multiplier */
  bsProb = {{"Equal"}};                                /* Prior probability: bias */

  if(nXcat < 2){
    usedXcat = 1;
  }else{
    usedXcat = nXcat$1;
```

```
}
h0group = {1, usedXcat}["_MATRIX_ELEMENT_COLUMN_*nBiasCat"];

if(nBiasCat < 2){
  beeClass = 2;
}else{
  beeClass = nBiasCat$1;
}
classOfBias = splitRate(beeClass, biasMaker[1], biasMaker[2]);
classOfBias[0] = smallestBiasValue;

h0description = h0clas[0];
classRule = (h0description != "Exact") && (h0description != "CutOff");
classRule = classRule || (Columns(h0clas)<2);
if(classRule){
  bugged = bugged + 1;
  fprintf(stdout, "\nError: A wrong description of the bias categories corr",
          "esponding to the null hypothesis was specified. Please adjust!\n");
}

if(h0description == "CutOff"){
  h0cutOff = h0clas[1];
  extend = +(classOfBias["_MATRIX_ELEMENT_VALUE_<=h0cutOff"]);
  h0size = Columns(h0group);
  need0 = {h0size, extend*h0size};
  need0 = need0["_MATRIX_ELEMENT_ROW_==(_MATRIX_ELEMENT_COLUMN_%h0size)"];
  need0 = h0group * need0;
  need0 = need0["_MATRIX_ELEMENT_VALUE_+(_MATRIX_ELEMENT_COLUMN_$h0size)"];
  h0group = need0;
}

if(h0description == "Exact"){
  userLength = Columns(h0clas) - 1;
  userNull = {1, userLength};
  for(picked=1; picked<Columns(h0clas); picked+=1){
    userNull[picked-1] = h0clas[picked];
  }
  userNull = sort(userNull);

  userGroup = {1, userLength};
  userTag = 0;
  for(null=0; null<beeClass; null+=1){
    if(userTag < userLength){
      searchedBias = userNull[userTag];
    }
    theBias = classOfBias[null];
    if(theBias == searchedBias){
      userGroup[userTag] = null;
      userTag = userTag + 1
```

```
    }
  }

  groupLength = userLength * Columns(h0group);
  h0temp = {1,groupLength};
  h0member = 0;
  for(multI=0; multI<Columns(h0group); multI+=1){
    xMember = h0group[multI];
    for(biasJ=0; biasJ<userLength; biasJ+=1){
      bMember = userGroup[biasJ];
      h0temp[h0member] = xMember + bMember;
      h0member = h0member + 1;
    }
  }
  h0group = h0temp;
}
fprintf(anStore+"nullClass.txt", CLEAR_FILE,
        "\nNull hypothesis rows:\n", h0group);

/* ><><>< Identify some misspecification errors ><><>< */
bfCheck = {1,3}["_MATRIX_ELEMENT_COLUMN_==bfMeth"];
bfCheck = +bfCheck;
if(!bfCheck){
  bugged = bugged + 1;
  fprintf(stdout, "\nError: An invalid prior odd computation index was spe",
          "cified. The 9th input can only be 0, 1 or 2!\n");
}

if(!bugged){
  /* ><><>< Conditional likelihood computation ><><>< */
  timeCount = Time(0);

  bTa = preLikelyCompute(seqSafe, luckyAa, biasMaker, xMaker, bsProb, mxProb,
                         topoSafe, fgTaxa, instantRmatrix, anStore, "False");

  timeCounted = Time(0) - timeCount;
  fprintf(stdout, "\nConditional likelihood computed in ",
          Eval( Format(timeCounted,0,4) ), " seconds.\n");

  fprintf(anStore+"betaMatrix.txt", CLEAR_FILE, "\n", bTa);
  fprintf(stdout, "\n");

  /* ><><>< Create posterior inference storages ><><>< */
  initialMaths = {};
  tilegets = Rows(bTa);

  if(meth[0]){
    temporaryGibbs = {};
    temporaryGibbsTime = {};
```

```
  temporaryGibbsFactor = {};
  gibbsBFhome = anStore + "hGibbsBFs.txt";
  gibbsTimeHome = anStore + "hGibbsCpuSecs.txt";
  gibbsMathHome = anStore + "hGibbsMatices.txt";
}
if(meth[1]){
  temporaryCvb = {};
  temporaryCvbTime = {};
  temporaryCvbFactor = {};
  cvbBFhome = anStore + "hCvbBFs.txt";
  cvbTimeHome = anStore + "hCvbCpuSecs.txt";
  cvbMathHome = anStore + "hCvbMatices.txt";
}
if(meth[2]){
  temporaryCvbZero = {};
  temporaryCvbZeroTime = {};
  temporaryCvbZeroFactor = {};
  cvbZeroBFhome = anStore + "hCvbZeroBFs.txt";
  cvbZeroTimeHome = anStore + "hCvbZeroCpuSecs.txt";
  cvbZeroMathHome = anStore + "hCvbZeroMatices.txt";
}
hepsilon = 10^(-20);
initializerHome = anStore + "hInitialGammas.txt";

for(mission=0; mission<Columns(tilegets); mission+=1){

  betaName = tilegets[mission];
  nobleBeta = bTa[betaName];

  /* ><><>< Specify other general initialization parameters ><><>< */
  Gama0 = initialGammas(nobleBeta);
  initialMaths[betaName] = Gama0;
  ratePrior = {1,Rows(nobleBeta)}["0.5"];
  fprintf(initializerHome, CLEAR_FILE, "\n", initialMaths);

  /* ><><>< Collapsed variational Bayes approximation ><><>< */
  if(meth[1]){

    if(bfMeth == 2){
      priorOmatC = p2param["CVB"];
    }

    oneTime = Time(0);
    cvBayes = cvbMethod(nobleBeta, hepsilon, ratePrior, Gama0,
                        anStore, "False");
    cvbBF = bayesFactor(cvBayes, h0group, bfMeth, priorOmatC);
    twoTime = Time(0) - oneTime;
    threeTime = Format(twoTime, 0, 4);
```

```
  temporaryCvbTime[betaName] = Eval(threeTime);
  temporaryCvbFactor[betaName] = cvbBF;
  temporaryCvb[betaName] = cvBayes;

  fprintf(cvbBFhome, CLEAR_FILE, "\n", temporaryCvbFactor);
  fprintf(cvbTimeHome, CLEAR_FILE, "\n", temporaryCvbTime);
  fprintf(cvbMathHome, CLEAR_FILE, "\n", temporaryCvb);
  fprintf(stdout, "\nCompleted CVB analysis.\n\n");
}

/* ><><>< Collapsed variational Bayes zero approximation ><><>< */
if(meth[2]){

  if(bfMeth == 2){
    priorOmatC0 = p2param["CVB0"];
  }

  oneTime = Time(0);
  cvb_0 = cvbZero(nobleBeta, hepsilon, ratePrior, Gama0,
                  anStore, "False");
  cvb0BF = bayesFactor(cvb_0, h0group, bfMeth, priorOmatC0);
  twoTime = Time(0) - oneTime;
  threeTime = Format(twoTime, 0, 4);

  temporaryCvbZero[betaName] = cvb_0;
  temporaryCvbZeroFactor[betaName] = cvb0BF;
  temporaryCvbZeroTime[betaName] = Eval(threeTime);

  fprintf(cvbZeroMathHome, CLEAR_FILE, "\n", temporaryCvbZero);
  fprintf(cvbZeroTimeHome, CLEAR_FILE, "\n", temporaryCvbZeroTime);
  fprintf(cvbZeroBFhome, CLEAR_FILE, "\n", temporaryCvbZeroFactor);
  fprintf(stdout, "\nCompleted CVB-0 analysis.\n\n");
}

/* ><><>< Collapsed Gibbs approximation ><><>< */
if(meth[0]){
  exploring = "False";                          /* Save all Gibbs' samples? */
  sampSize = 01000;                                  /* desired sample size */
  burnIn = 05000;                                           /* burn-in size */
  thin = 00020;                                                /* thin size */

  if(bfMeth == 2){
    priorOmatG = p2param["Gibbs"];
  }

  oneTime = Time(0);
  geebsMath= collapsedGibbs(sampSize, thin, burnIn, nobleBeta,
                ratePrior, Gama0, h0group, bfMeth, priorOmatG,
                exploring, anStore, "False");
```

```
        geebsBF = bayesFactor(geebsMath, h0group, bfMeth, priorOmatG);
        twoTime = Time(0) - oneTime;
        threeTime = Format(twoTime, 0, 4);

        temporaryGibbs[betaName] = geebsMath;
        temporaryGibbsFactor[betaName] = geebsBF;
        temporaryGibbsTime[betaName] = Eval(threeTime);

        fprintf(gibbsMathHome, CLEAR_FILE, temporaryGibbs);
        fprintf(gibbsTimeHome, CLEAR_FILE, temporaryGibbsTime);
        fprintf(gibbsBFhome, CLEAR_FILE, temporaryGibbsFactor);
        fprintf(stdout, "\n\nCompleted Gibbs analysis.\n\n");
      }
    }
  }
  return 0;
}
```

Example

```
line0 = ">S1\nATTCT\n>S2\nTTACT\n>S3\nGTTCA\n>S4\nGTTCA";
line1 = "\n>S5\nTTACA\n>S6\nTTACC\n>S7\nATTGT\n>S8\nTTACA";
dtLines = "" + line0 + line1;
DataSet testSeq = ReadFromString(dtLines);
DataSetFilter testSeqF = CreateFilter(testSeq, 1);
fprintf("testSeq.txt", CLEAR_FILE, testSeqF);

sometin1 = "((((S1:1,S2:1)iN0:1,(S3:1,S4:1)iN1:1)iN4:1,";
sometin2 = "((S5:1,S6:1)iN2:1,(S7:1,S8:1)iN3:1)iN5:1)iN6:1)";
topos = "" + sometin1 + sometin2;
fprintf("testTopo.txt", CLEAR_FILE, topos);

nState =04;
omgSize = 04;
buySize = 04;
tRes = "Identify";
approach = {{1,1,1}};
fdTaxa = {{"S1","S7"}};
dataSafe = "testSeq.txt";
toupology = "testTopo.txt";
bMat = {nState,nState}["1"];
h0thresh = {{"Exact",1,0.5}};
allRateCat = omgSize * buySize;
pOddM = {allRateCat,testSeqF.sites}["1/allRateCat"];
p2offering = {"Gibbs":pOddM,"CVB0":pOddM,"CVB":pOddM};

tFADE (dataSafe, buySize, omgSize, fdTaxa, h0thresh,
       tRes, toupology, approach, 02, bMat, "", p2offering);
```

simAnalysisInterface.bf	Simulation analysis interface

Description

This HyPhy batch file provides an interface for performing simulation analysis of the topic model-based approach to inferring episodic directional selection in protein coding sequences. This script should be called from the folder that contains all the fifteen previously described batch files. If more parameters, than the ones allowed for here, then the batch file where the affected function was defined may be consulted.

Usage

N/A

Arguments

```
/* ><>< ================================================================== ><>< */
/*                      THIS SECTION MAY BE ALTERED BY THE USER.                 */
/* ><>< ================================================================== ><>< */

nState = 20;                                                        /* Amino acid */
nLineage = 016;                                                   /* Specie size */
siteSize = 0020;                                               /* Number of sites */
outputBank = "";                                             /* Output folder path */
byasClasses = 08;                                            /* Size of bias class */
likelyXclass = 00;                                        /* Multiplier categories */
fgFraction = 0.25;                                        /* Foreground proportion */
clusterForegd = 0;                                  /* Non-spaced foreground taxa? */
pryorOddMethod = 1;                                 /* Prior odd estimation method */
byasParameter = 1000;                               /* Simulation's bias parameter */
approach = {{1,1,1}};                               /* Use {{Gibbs?, CVB?, CVB0?}} */
stem = {{0.05, 0.05}};                            /* Range for branch length draws */
simulationX = {{000}};                          /* Simulation multiplier parameter */
calcTarget = "Identify";                       /* Target handling during inference */
h0Xters = {{"Exact", 0}};                      /* Null hypothesis class characters */
optimizeBranchLength = 0;                   /* Topology contains optimized lengths? */
nullSitesProportion = 0.9;                  /* Create data with 0 bias in this much */
simTarget = "fixedTarget:E";                /* Description of target for simulation */
iBaseMat = {nState,nState}["1"];            /* Specify baseline substitution matrix */
eqOccur = {1, nState}["1/nState"];         /* Stationary frequencies for simulation */
```

Value

See the included batch files.

Function

```
/* ><>< ================================================================== ><>< */
/*              THIS SECTION DOES NOT NEED TO BE ALTERED BY THE USER.             */
/* ><>< ================================================================== ><>< */

 mixVector = {{nullSitesProportion, byasParameter}};
 biasInfo0 = {{simTarget, clusterForegd}};
 biasInfo1 = {{fgFraction}};

 targetResidues = mixedBiasData(mixVector, siteSize, nLineage, stem, biasInfo1,
                                biasInfo0, simulationX, iBaseMat, eqOccur,
                                outputBank);

 topoBank = outputBank + "simulationTopology.txt";      /* Tree topology location */
 topoBank0 = outputBank + "recycledTopology1.txt";
 topoBank1 = outputBank + "recycledTopology2.txt";
 fscanf(topoBank, "Raw", simsTopology);
 fprintf(topoBank0, CLEAR_FILE, simsTopology);
 fprintf(topoBank1, CLEAR_FILE, simsTopology);
 topoBank2 = {{topoBank0, optimizeBranchLength}};

 dataSafe = outputBank + "simulatedData.txt";           /* Sequence data location */
 agedSafe = outputBank + "simulatedAncestorData.txt";   /* Ancestor data location */
 foreHouse = outputBank + "biasedTaxa.txt";               /* Foreground taxa path */
 fscanf(foreHouse, "Matrix", fgIndex);                     /* Foreground branches */

 /* ><>< Infer episodic directional selection ><>< */
 noUse = tFADE (dataSafe, byasClasses, likelyXclass, fgIndex, h0Xters,
                calcTarget, topoBank2, approach, pryorOddMethod,
                iBaseMat, outputBank, nothing);

 /* ><>< Generate replica of Table 5. in Murrell et. al. (2012) ><>< */
 theTable = countFavours(dataSafe, agedSafe, topoBank1, fgIndex,
                         targetResidues, outputBank);
```

Example

N/A

`tFadeInterface.bf` Interface for implementing topic model EDS analysis

Description

The purpose of this HyPhy batch file is to allow for easy implementation of `tFADE`, a FUBAR and topic modelling algorithm inspired function that is useful

for model-based episodic directional selection inference. The idea is to protect the user from being overwhelmed by long lines of code in the `topicFade.bf` file where the function was defined. Details of how the function works can be found in the main script.

Usage

```
tFADE (dataSafe, buySize, omgSize, fdTaxa, h0thresh, tRes,
           toupology, approach, pOddMeth, bMat, outs, nothing)
```

Arguments

See `topicFade.bf`.

Value

See `topicFade.bf`.

Function

```
#include "topicFade.bf";
```

Example

```
nState = 4;
omgSize = 04;                 /* Number of multiplier categories to be used */
buySize = 04;                       /* Number of bias categories to be used */
pOddMeth = 1;                       /* Prior odd computation method's index */
outs = "folder/";                   /* The path to a folder for the outputs */
tRes = "Explore";                   /* Details of target residue to analyze */
approach = {{1,1,1}};               /* Method to use: {{Gibbs?,CVB?,CVB0?}} */
fdTaxa = {{"stem1","stem2"}};              /* Foreground branches statement */
bMat = {nState,nState}["1"];                    /* Base substitution matrix */
statFreq = {1, nState}["1"];                     /* Equilibrium frequencies */
h0thresh = {{"CutOff",1e-04}};                    /* H0 category definition */
dataSafe = "folder/filename.fileExtension";        /* Path to sequence file */
toupology = "folder/filename.fileExtension";       /* Path to topology file */

tFADE (dataSafe, buySize, omgSize, fdTaxa, h0thresh, tRes,
       toupology, approach, pOddMeth, bMat, outs, nothing);
```

B.2 R code

topicFade.R	Topic model-based approach to inferring EDS

Description

The primary function, `tFade`, that is described here is a complementary `R` version of `topicFade.bf`. It may be used to infer episodic directional selection in protein sequences using the topic model-based algorithms that were developed in this dissertation.

Usage

```
tFADE (bta, alfa, init_gama, h0Class, preOddMeth=1, priorP=c(),
     meth=c(1,1,1), burn.in=5000, thin.size=20, samp.size=1000,
     threshold=1e-20, saveAll=F)
```

Arguments

`bta`: A conditional likelihood matrix, over pre-specified rate categories, computed from the sequence alignment for which inference is sought. The dimension of the matrix is expected to be (number of rate categories) $\times$ (number of sites).

`alfa`: A vector that contains the prior probabilities of the different rate categories.

`init_gama`: A matrix of similar dimension to the given conditional probability matrix, `bta`. It should give the initial probabilities to be used to initialise the inference process.

`h0Class`: A vector that states the indices of the rows of `bta` that correspond to the null hypothesis categories.

`preOddMeth`: A scalar that must be an element of {0,1,2}. It indicates the method to adopt when computing the prior odd for the site-specific Bayes factors. If `preOddMeth` is zero (default), prior odd will be computed as the number of the rows of `bta` attached to the alternative hypothesis, divided by the number of rows related to the null hypothesis. If `preOddMeth=1`, prior odd will computed as the sum of all the posterior probabilities in the rows of the alternative hypothesis related classes across all the matrix columns, divided by the sum of all the posterior probabilities in all the rows related to the null hypothesis categories across all columns. If `preOddMeth=2`, similar approach to the case described for `preOddMeth=1` will be used except that the posterior probabilities will be replaced by the prior probabilities supplied as `priorP`.

`priorP:` A matrix of prior probabilities that is relevant only when `preOddMeth=2`.

`meth:` An indicator vector of three elements. Each element should be `0` or `1`. It indicates which of collapsed Gibbs sampling (CGS), collapsed variational Bayes (CVB) and zero-order collapsed variational Bayes (CVB0) algorithms should be implemented. Its first correspnds to CGS, its second element is associated with CVB, while its third element is for CVB0. For example, specifying `0` as the first element of the vector will cause CGS not to be activated. The default is `c(1,1,1)`.

`burn.in:` A scalar that states the preferred burn-in size. It is only relevant when CGS is activated. The default is $5,000$.

`thin.size:` A scalar that states the preferred thin size. It is only relevant if CGS is activated. The default is 20.

`samp.size:` A scalar that states the preferred final sample size after burning-in and thinning the CGS draws. It is only relevant if CGS is activated. The default is $1,000$.

`threshold:` A scalar that indicates when the variational Bayes algorithms should be terminated. The concerned algorithms only teminantes if the maximum element in the absolute difference between the mth and the $(m-1)$th posterior probability matrix is less than the provided value. After the $1,000$th iteration, the algorithms are set to terminate regardless of whether the threshold has been met. It is only relevant if CVB or CVB0 is activated. The default is 10^{-20}.

`saveAll:` A `TRUE` or `FALSE`(default) statement on whether every Gibbs Bayes factor drawn should be saved.

Value

After a successful implementation, distinctly named text files of the sought matrix of posterior probabilities and Bayes factors will be saved in the working directory for the different approaches implemented. For CGS, standard deviations will also be reported.

Function

```
# ~~~~~~~~~~~~~~~~ #
# --- Preamble --- #
# ~~~~~~~~~~~~~~~~ #
 rm(list=ls())

 require("foreach")
 require("doParallel")
```

```
 registerDoParallel()
 #options(digits=19)

# ~~~~~~~~~~~~~~~~~~~ #
# --- Normalization --- #
# ~~~~~~~~~~~~~~~~~~~ #
normalize <- function(vect, K){
  temp.mat <- matrix(vect, nrow=K)
  sums <- colSums(temp.mat)
  normal <- temp.mat %*% diag(1/sums)
  as.vector(normal)
}

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ #
# Posterior probability initiation #
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ #
initialGamma <- function(conditionLikely){
  nRates <- nrow(conditionLikely)
  numerator <- rowSums(conditionLikely)
  denominantor <- colSums(conditionLikely)
  qty <- length(numerator)
  iGamma <- sapply(denominantor, function(a) (numerator/a)*runif(qty,0.8,1.2))
  if(min(iGamma)==0){
    iGamma <- iGamma + matrix(runif(length(iGamma),0.5,1), qty)
  }
  iGamma <- normalize(as.vector(iGamma), nRates)
  iGamma <- matrix(iGamma, nRates)
  return(iGamma)
}

# ~~~~~~~~~~~~~~~~~~ #
# --- Bayes factor --- #
# ~~~~~~~~~~~~~~~~~~ #
bayesFactor <- function(postP, nullClass, preOddMeth=0, priorP=NULL){
  catSize <- nrow(postP)
  indicator <- sapply(1:catSize, function(a){sum(nullClass==a)})
  postP <- normalize(as.vector(postP),catSize)
  postP <- matrix(postP, catSize)

  h0post <- indicator %*% postP
  h0post <- sapply(h0post, function(a){a+((a==0)*(1e-300))})
  h1post <- 1 - h0post
  postOdds <- h1post / h0post

  if(preOddMeth==0){
    h0prior <- sum(indicator)
    h1prior <- catSize - h0prior
    priorOdd <- h1prior / h0prior
  }
```

```
  if(preOddMeth==1){
    priorOdd <- sum(h1post) / sum(h0post)
  }

  if(preOddMeth==2){
    print("Generating prior probability matrix.")
    priorP <- as.matrix(priorP)
    priorP <- normalize(as.vector(priorP),catSize)
    priorP <- matrix(priorP, catSize)
    h0prior <- indicator %*% priorP
    h1prior <- 1 - h0prior
    priorOdd <- sum(h1prior) / sum(h0prior)
  }

  bayesF <- postOdds / priorOdd
  return(bayesF)
}

# ~~~~~~~~~~~~ #
# --- Gibbs --- #
# ~~~~~~~~~~~~ #
gibs.func <- function(bta, alfa, init_gama, h0Class, preOddMeth, priorP,
                      burn.in, thin.size, samp.size, saveAll)
{
  bta <- as.matrix(bta)
  K <- nrow(bta)
  N <- ncol(bta)
  alfa.vect <- rep(alfa, N)
  indices <- data.frame(K.vect=rep(1:K, N), X.vect=rep(1:N, each=K))
  bta.vect <- as.vector(bta)

  est.z <- function(gama, N){
    mat.gama <- matrix(gama, ncol=N)
    needed <- foreach(i=1:N, .combine='c') %dopar% {
                sample(nrow(mat.gama), 1, prob=mat.gama[,i])
              }
    return(needed)
  }

  if(saveAll){
    fileName <- paste("rAllGibbsBFs.txt", sep="")
    write.table(NULL, fileName, row.names=F, col.names=F, quote=F)
  }

  store.index <- burn.in + seq(thin.size,(thin.size*samp.size),thin.size) + 1
  sumPostP <- sumSqPostP <- sumPostPbf <- sumSqPostPbf <- count <- 0
  c.index <- store.index[1]
  new.gama <- init_gama
```

```
superCount <- 1
not.enough <- T
while(not.enough){
  count <- count + 1
  updated.z <- est.z(new.gama, N)
  n.minus.i <- foreach(i=1:(N*K), .combine='c') %dopar% {
                 sum(updated.z[-indices[i,2]] == indices[i,1])
               }
  new.gama <- (alfa.vect + n.minus.i) * bta.vect
  gibbs.gama <- normalize(new.gama, K)
  newBfs <- bayesFactor(matrix(gibbs.gama,nrow=K), h0Class,
                        preOddMeth, priorP)

  if(saveAll){
    write.table(t(newBfs), fileName, row.names=F, col.names=F,
                quote=F, append=T)
  }

  if(count == c.index){
    sumSqPostP <- sumSqPostP + (gibbs.gama^2)
    sumPostP <- sumPostP + gibbs.gama

    sumSqPostPbf <- sumSqPostPbf + (newBfs^2)
    sumPostPbf <- sumPostPbf + newBfs

    superCount <- superCount + 1
    c.index <- store.index[superCount]
  }

  print(sprintf("Obs %.0f of %.0f for K=%.0f, N=%.0f",
      count, max(store.index), K, N))
  not.enough <- count < max(store.index)
}
rGamma <- (1/samp.size) * sumPostP
rGamma <- matrix(normalize(rGamma,K), nrow=K)

rGammaBF <- (1/samp.size) * sumPostPbf

rGammaSd <- sumSqPostP - ((1/samp.size) * (sumPostP^2))
rGammaSd <- sqrt( (1/(samp.size-1)) * rGammaSd )
rGammaSd <- matrix(rGammaSd, nrow=K)

rGammaSdBF <- sumSqPostPbf - ((1/samp.size) * (sumPostPbf^2))
rGammaSdBF <- sqrt( (1/(samp.size-1)) * rGammaSdBF )

colnames(rGamma) <- colnames(rGammaSd) <- paste("Site", 1:N, sep="")
names(rGammaBF) <- names(rGammaSdBF) <- paste("Site", 1:N, sep="")
write.table(rGamma, "rGibbsPostProb.txt", row.names=F, col.names=F)
write.table(rGammaSd, "rGibbsPpSd.txt", row.names=F, col.names=F)
```

```
  write.table(t(rGammaBF), "rGibbsBFs.txt", row.names=F, col.names=F)
  write.table(t(rGammaSdBF), "rGibbsBFsSd.txt", row.names=F, col.names=F)

  print("Completed Gibbs sampling procedure.")
}

# ~~~~~~~~~~~~ #
# --- CVBI --- #
# ~~~~~~~~~~~~ #
cvbi.func <- function(bta, alfa, init_gama, threshold){
  bta <- as.matrix(bta)
  K <- nrow(bta)
  N <- ncol(bta)
  workingBta <- t(bta)

  nsims <- 0
  new.gama <- t(init_gama)
  not.converge <- T
  while(not.converge){
    nsims <- nsims + 1
    old.gama <- new.gama

    part1 <- colSums(new.gama)
    part1 <- rep(1,N) %*% rbind(part1)
    part1 <- part1 - new.gama
    part1 <- part1 + (rep(1,N) %*% rbind(alfa))

    parta <- new.gama * (1-new.gama)
    part2 <- colSums(parta)
    part2 <- rep(1,N) %*% rbind(part2)
    part2 <- part2 - parta
    part2 <- part2 / (2 * (part1*part1))

    new.gama <- exp(-part2)
    new.gama <- workingBta * part1 * new.gama
    new.gama <- t(new.gama)
    new.gama <- normalize(as.vector(new.gama), K)
    new.gama <- matrix(new.gama, K)
    new.gama <- t(new.gama)

    differ <- abs(old.gama - new.gama)
    criterion <- max(differ)

    print(sprintf("Max. diff. after %.0f run(s) = %.25f for K=%.0f, N=%.0f",
                  nsims, criterion, K, N))

    not.converge <- (criterion >= threshold) & (nsims < 1000)
  }
  return(t(new.gama))
```

```
}

# ~~~~~~~~~~~~ #
# --- CVB0 --- #
# ~~~~~~~~~~~~ #
cvb0.func <- function(bta, alfa, init_gama, threshold){
  bta <- as.matrix(bta)
  K <- nrow(bta)
  N <- ncol(bta)

  workingBta <- t(bta)
  nsims <- 0
  new.gama <- t(init_gama)
  not.converge <- T
  while(not.converge){
    nsims <- nsims + 1
    old.gama <- new.gama

    part1 <- colSums(new.gama)
    part1 <- rep(1,N) %*% rbind(part1)
    part1 <- part1 - new.gama
    part1 <- part1 + (rep(1,N) %*% rbind(alfa))

    new.gama <- workingBta * part1
    new.gama <- t(new.gama)
    new.gama <- normalize(as.vector(new.gama), K)
    new.gama <- matrix(new.gama, K)
    new.gama <- t(new.gama)

    differ <- abs(old.gama - new.gama)
    criterion <- max(differ)

    print(sprintf("Max. diff. after %.0f run(s) = %.25f for K=%.0f, N=%.0f",
                  nsims, criterion, K, N))

    not.converge <- (criterion >= threshold) & (nsims < 1000)
  }
  return(t(new.gama))
}

# ~~~~~~~~~~~~~~~~~ #
# --- Topic FADE --- #
# ~~~~~~~~~~~~~~~~~ #
tFADE <- function(bta, alfa, init_gama, h0Class, preOddMeth=0, priorP=c(),
                  meth=c(1,1,1), burn.in=5000, thin.size=20, samp.size=1000,
                  threshold=1e-20, saveAll=F)
{
  useGibbs <- meth[1]
  useCvbi <- meth[2]
```

```
  useCvb0 <- meth[3]

  if(useCvbi){
    cvbiTime <- as.numeric(proc.time()["elapsed"])

    analyseCi <- cvbi.func(bta, alfa, init_gama, threshold)
    analyseCi <- matrix(analyseCi, nrow(bta))
    probsOdds <- bayesFactor(analyseCi, h0Class, preOddMeth, priorP)
    colnames(analyseCi) <- paste("Site", 1:ncol(analyseCi), sep="")
    write.table(analyseCi, "rCvbiPostProb.txt", row.names=F, col.names=F)
    write.table(t(probsOdds), "rCvbBFs.txt", row.names=F, col.names=F, quote=F)

    cvbiTime <- as.numeric(proc.time()["elapsed"]) - cvbiTime
    write.table(t(cvbiTime), "rCvbTime.txt", row.names=F, col.names=F, quote=F)
    print("Completed collapsed variational Bayes' analysis.")
  }

  if(useCvb0){
    cvb0Time <- as.numeric(proc.time()["elapsed"])

    analyseC0 <- cvb0.func(bta, alfa, init_gama, threshold)
    analyseC0 <- matrix(analyseC0, nrow(bta))
    probsOdd0 <- bayesFactor(analyseC0, h0Class, preOddMeth, priorP)
    colnames(analyseC0) <- paste("Site", 1:ncol(analyseC0), sep="")
    write.table(analyseC0, "rCvb0PostProb.txt", row.names=F, col.names=F)
    write.table(t(probsOdd0), "rCvb0BFs.txt", row.names=F, col.names=F,
                quote=F)

    cvb0Time <- as.numeric(proc.time()["elapsed"]) - cvb0Time
    write.table(t(cvb0Time), "rCvb0Time.txt", row.names=F, col.names=F, quote=F)
    print("Completed implementation of collapsed variational Bayes-Zero.")
  }

  if(useGibbs){
    gibbsTime <- as.numeric(proc.time()["elapsed"])

    analyzeGs <- gibs.func(bta, alfa, init_gama, h0Class, preOddMeth, priorP,
                           burn.in, thin.size, samp.size, saveAll)

    gibbsTime <- as.numeric(proc.time()["elapsed"]) - gibbsTime
    write.table(t(gibbsTime), "rGibbsTime.txt", row.names=F,
                col.names=F, quote=F)
  }
  print("Completed assigned topic FADE analysis.")
}

# ~~~~~~~~~~~~~~~ #
# Code ends here. #
# ~~~~~~~~~~~~~~~ #
```

Example

```
analysisData <- rbind(
  c(0.000038133,0.005280549,0.000252946,0.000004776,0.000095518),
  c(0.000061862,0.004688458,0.000361064,0.000006283,0.000131391),
  c(0.000094077,0.004074869,0.000487594,0.000007960,0.000171503),
  c(0.000864087,0.000000001,0.001534776,0.000001926,0.000453832))

priorPmatrix <- matrix(1, nrow(analysisData), ncol(analysisData))
initialLikely <- matrix(1, nrow(analysisData), ncol(analysisData))
alfa <- rep(0.5, nrow(analysisData))
approach <- c(1,1,1)
exploratory <- F
cutOff <- 1e-20
allSsize <- 150
bInSize <- 100
tInSize <- 10
h0Klass <- 1
preOdd <- 2

useFade <- tFADE(analysisData, alfa, initialLikely, h0Klass, preOdd,
                 priorPmatrix, approach, bInSize, tInSize, allSsize,
                 cutOff, exploratory)
useFade
```

---

tFadeInterface.R Interface for implementing topic model EDS analysis

---

Description

The purpose of this R script is to allow for easy implementation of `tFADE`, a FUBAR and topic modelling algorithm inspired function that is useful for model-based episodic directional selection inference. The idea is to protect the user from being overwhelmed by long lines of code in the topicFade.R script where the function was defined. Details of how the function works can be found in the main script.

Usage

```
tFADE (bta, alfa, init_gama, h0Class, preOddMeth=1, priorP=c(),
     meth=c(1,1,1), burn.in=5000, thin.size=20, samp.size=1000,
     threshold=1e-20, saveAll=F)
```

Arguments

See topicFade.R.

Value

See topicFade.R.

Function

```
source("topicFade.R")
```

Example

```
# --- Conditional matrix needed to be analyzed --- #
analysisData <- read.csv("~/Desktop/r/01-output/betaMatrix.txt", header=F)

# --- Prior probability matrix to be used if prior odd approach equals 2 --- #
priorPmatrix <- 0

# --- Prior distribution of sequence over rate categories --- #
alfa <- rep(0.5, nrow(analysisData))

# --- Indicate the approximation methods to use (Gibbs?, CVB?, CVB0?) --- #
approach <- c(1,1,1)

# --- Must all Gibbs drawn samples be saved? --- #
exploratory <- TRUE

# --- Degree of error tolerance for CVB and CVB0 --- #
cutOff <- 1e-20

# --- Required Gibbs sample size after thinning and burning --- #
allSsize <- 10000

# --- Burning size for Gibbs approach --- #
bInSize <- 000

# --- Thinning size for Gibbs approach --- #
tInSize <- 1

# --- Indices of the rows of H0 in the conditional likelihood matrix --- #
h0Klass <- 1

# --- Which prior odd computation approach should be used? --- #
preOdd <- 1

# --- Initial conditional matrix for initializing the approx. process --- #
initialLikely <- initialGamma(analysisData)
```

```
initialBFs <- bayesFactor(initialLikely, h0Klass, preOdd)
write.table(t(initialBFs), "rInitialBFs.txt", row.names=F, col.names=F)
write.table(initialLikely, "rInitialLikely.txt", row.names=F, col.names=F)

# --- Topic FADE function --- #
useFade <- tFADE(analysisData, alfa, initialLikely, h0Klass, preOdd,
                 priorPmatrix, approach, bInSize, tInSize, allSsize,
                 cutOff, exploratory)
useFade
```