Statistical Applications in Genetics and Molecular Biology

Volume 11, *Issue* 4 2012 *Article* 14

Hessian Calculation for Phylogenetic Likelihood based on the Pruning Algorithm and its Applications

Toby Kenney, *Dalhousie University*
Hong Gu, *Dalhousie University*

Recommended Citation:
Kenney, Toby and Gu, Hong (2012) "Hessian Calculation for Phylogenetic Likelihood based on the Pruning Algorithm and its Applications," *Statistical Applications in Genetics and Molecular Biology*: Vol. 11: Iss. 4, Article 14.

Hessian Calculation for Phylogenetic Likelihood based on the Pruning Algorithm and its Applications

Toby Kenney and Hong Gu

Abstract

We analytically derive the first and second derivatives of the likelihood in maximum likelihood methods for phylogeny. These results enable the Newton-Raphson method to be used for maximising likelihood, which is important because there is a need for faster methods for optimisation of parameters in maximum likelihood methods. Furthermore, the calculation of the Hessian matrix also opens up possibilities for standard likelihood theory to be applied, for inference in phylogeny and for model selection problems. Another application of the Hessian matrix is local influence analysis, which can be used for detecting a number of biologically interesting phenomena. The pruning algorithm has been used to speed up computation of likelihoods for a tree. We explain how it can be used to speed up the computation for the first and second derivatives of the likelihood with respect to branch lengths and other parameters. The results in this paper apply not only to bifurcating trees, but also to general multifurcating trees. We demonstrate the use of our Hessian calculation for the three applications listed above, and compare with existing methods for those applications.

KEYWORDS: phylogeny, likelihood, Newton-Raphson, Markov process, Hessian

1 Introduction

Maximum likelihood methods for phylogeny are becoming more popular, as computers become more powerful, allowing larger data sets and more complicated models to be used for phylogenetic inference. Finding the maximum likelihood estimate (MLE) is a numerical optimisation problem. The most popular method for this type of numerical optimisation is the Newton-Raphson method. However, the Newton-Raphson method requires the first and second derivatives of the objective function (in this case, the likelihood). It is therefore necessary to be able to calculate these derivatives, or to numerically approximate them. Numerically approximating the first derivatives is common practice in the currently existing Newton-based algorithms in phylogenetic analysis. The basic idea is to approximate the tangent of the curve by using a chord. As the length of the chord gets shorter, its slope will tend to the derivative. By choosing a sufficiently short chord, a good approximation of the first derivative can be obtained. In principal the derivative of the derivative can be approximated using the same method. However numerically approximating second derivatives in this naive way usually would not provide the accuracy and stability needed, because it involves taking the difference of two function values that are very close, which greatly increases rounding errors. This is particularly true when the first derivative is nearly zero. It also does not have any advantage in computational complexity over the analytical calculation provided here, since it requires calculating the likelihood at $O((b+p)^2)$ points, where b is the number of branches of the tree and p is the number of all other parameters. An alternative is to use more sophisticated quasi-Newton methods, which build up an approximation to the Hessian matrix during the optimisation process. These methods are used in a number of phylogeny software packages, such as PAML, NHML and PAUP*. However, it is widely acknowledged that the lack of "an efficient and accurate algorithm for optimising the parameters" is a "major difficulty with likelihood based inference" (Bryant, Galtier and Poursat, 2005). We hope that the Hessian calculation presented in this paper will help to remedy this situation.

Besides being useful in the Newton-Raphson optimisation method, the second derivative or Hessian matrix is fundamentally important in statistical theory for inference. The Fisher information matrix is important for a number of aspects of statistical inference, such as the Cramér-Rao bound for the variance of an unbiased estimator, and for the score test in hypothesis testing (see e.g. Bickel and Doksum, 2001). In particular, based on the likelihood theory, under certain regularity conditions the maximum likelihood estimate is asymptotically normally distributed around its true value when the sample size tends to infinity, with the asymptotic variance given by the inverse of the Fisher information matrix (also see Bickel and Doksum, 2001). In practice, the Fisher information matrix often cannot be calcu-

lated, and the most convenient estimator of it is the sample average of the outer product of the scores (the first order derivative) or the negative Hessian matrix of the likelihood function evaluated at the maximum likelihood estimate. The negative Hessian is also called the observed information. The estimator based on the scores is usually easier to calculate but less efficient compared to the variance estimator based on Hessian matrix (Porter, 2002). Efron and Hinkley (1978) also showed that the observed information can sometimes even be preferable to the Fisher information in real data analysis.

Statistical models represent our assumptions about the approximate mechanism with which the data are generated. Following the specified model and estimation of the parameters in the model, it is always important and interesting to explore whether the analysis results are sensitive to the model and/or data. With the Hessian matrix available, the local influence method proposed by Cook (1986) can be applied in the sensitivity analysis of any model parameters for a fixed tree topology. Sensitivity analysis to data perturbation reveals outliers to the specified model.

The first derivatives of likelihood with respect to branch lengths or parameters affecting the rate matrix are known (Schadt, Sinsheimer and Lange, 1998, Schadt and Lange, 2002). Furthermore, second derivatives with respect to a single branch length are easy, and have been used at least as early as Kishino, Miyata and Hasegawa (1990). The second derivatives with respect to two different branch lengths are also known — Bryant, Galtier and Poursat (2005) mention a modification of the pruning algorithm to compute the gradient and Hessian with respect to branch lengths, but do not give a reference, and this modification does not appear to be widely known. However, despite the importance of the second derivatives, second derivatives involving model parameters are not calculated for the likelihoods arising from phylogeny. In this article, we develop a method for calculating second derivatives with respect to parameters that affect the rate matrix, and also branch lengths. The results in this paper apply not only to bifurcating trees, but also to general multifurcating trees.

We will first present the analytical results for the derivatives of phylogenetic likelihood and use an example throughout to illustrate how to calculate the derivatives through a tree traversal algorithm in Section 2. We then outline the algorithm and complexity of the algorithm in Section 3. This section provides the details that allow a programmer to implement the results. Then in Section 4, we present three different applications of the Hessian matrix. First we provide a comparison of the computation speeds using the Newton-Raphson method both for our exact Hessian calculation and for the approximation using the scores, with PAML (Yang, 2007), which uses quasi-Newton methods, on six real data sets with a variety of numbers of taxa and sequence lengths. We then provide a comparison of the confidence intervals built using likelihood theory with those obtained using a non-parametric

bootstrap on simulated data. Finally, we apply local influence analysis, which is computed using the Hessian matrix, to the problem of detecting sites under positive selection in codon models. Finally some concluding remarks and discussion are given.

2 Theory

There are two essential components in our methods of calculating the derivatives of phylogenetic likelihood. The first is the analytical solution of the first and second derivatives of the transition matrix with respect to any parameter which influences it, through which the derivatives of likelihood on each branch are calculated. The second component is a tree traversal algorithm which efficiently transforms the derivatives calculated on each branch into derivatives of the whole tree likelihood function. This algorithm is based on the pruning algorithm, which is an efficient way to compute the likelihood for a given phylogeny (Felsenstein, 1973, 1981).

The data we are analysing consists of aligned DNA sequences — one sequence for each taxon in the tree. We can arrange the data as a matrix, where the rows are sequences, and the columns are sites. Thus, a site is one position in the (nucleotide, amino acid or codon) alignment. Under the assumption that the evolutionary process at each site is independent or conditionally independent given the parameters in the substitution model, the likelihood for a given tree is the product of the site likelihoods. Therefore, the derivatives can be worked out using the product rule, so the problem reduces to calculating the derivatives at each site. If we are considering the log-likelihood instead, then the derivative is the sum of the derivatives at each site. Of course, we can convert between the derivative of likelihood and the derivative of log likelihood easily — if $l = \log(L)$ then $\frac{\partial l}{\partial \beta} = \frac{1}{L}\frac{\partial L}{\partial \beta}$. For likelihoods in phylogeny, it is usually easiest to deal with the likelihoods for each site, and then convert to log likelihoods so that they can be added for different sites. Thus we will focus our discussion on the derivatives at one site.

For each branch length parameter which only affects one branch, calculating the derivative for each site is easy. For the other parameters, which affect every branch, we start by considering the heterotachy case, where the parameter takes a different value on each branch. We will refer to the value of a parameter β on a specific branch e as a branch parameter, and denote it β_e. In particular, the branch length of e is denoted as t_e. From the derivatives with respect to the branch parameters, we will be able to calculate the derivatives for the homotachy case, where β is a Q-matrix parameter. More precisely, the homotachy case can be viewed as a special case of the heterotachy model, where the parameter is constrained to have the same value on all branches, i.e. $\beta_e = \beta_{e'}$ for each pair of edges e and e'. This means

that the derivative is obtained via the chain rule as the sum of the derivatives with respect to all the branch parameters, i.e

$$\frac{\partial L}{\partial \beta} = \sum_{e} \frac{\partial L}{\partial \beta_e}$$

$$\frac{\partial^2 L}{\partial \beta \partial \gamma} = \sum_{e} \sum_{e'} \frac{\partial^2 L}{\partial \beta_e \partial \gamma_{e'}}$$

where β and γ may mean the same parameter or two different parameters and e and e' may be the same or different branches. The calculations of the above derivatives rely on the first and second derivatives of the transition matrix P on the edges e or e' with respect to any parameter which influences it.

We will first outline the model assumptions used in our methods and give some brief discussion about their importance. We then present the theoretical solution to differentiating the P matrix. To better present the tree traversal algorithm, we begin with a recap of the pruning algorithm followed by an illustrative example. Then we describe the solutions for the first and second derivatives in the following sections, with the same example being used to illustrate the calculation. Finally we end the *Theory* section with the extension of our methods to mixed models.

2.1 Model Assumptions

Most likelihood based phylogenetic models assume the evolutionary processes are Markov processes with the evolutionary relationships represented by trees. The models that implement the general time-reversible (GTR) model (Lanave et al, 1984) further assume that the sequences have evolved under globally stationary, reversible, and homogeneous conditions. Detailed definitions of these assumptions can be found in Jayaswal et al. (2005) and Ababneh et al. (2006), some scenarios that relax these global assumptions are also discussed in Jayaswal et al (2010). The global homogeneity mentioned in these papers mainly refers to time-homogeneity of the process, which is called homotachy below.

The methods developed here are under the assumptions that sites evolve independently and each site evolves under the GTR model assumptions. However some assumptions are more crucial and others are easily relaxed. We will list the assumptions made for our method, with some discussion of which assumptions are merely for convenience and can easily be relaxed, and which assumptions are more fundamental.

The general assumption about the models in our method is that the evolutionary process along each branch is a continuous-time homogeneous Markov chain. That is, for a given site, and a given branch of the tree, the transition probability is given

by $P(t) = e^{Qt}$, where Q is the instantaneous rate matrix, and is scaled so that the average rate of substitution at equilibrium equals 1. Thus the tree branch length t represents the expected number of substitutions per site. Note that the branch length t is not necessarily a linear transformation of the time. If the overall rate of change is not constant in time but the relative rate of different changes are constant, a non-linear scaling of time can be applied so that changes in the overall evolutionary rate along a branch are negated.

Among other assumptions made, many are purely for convenience of calculation and to increase the efficiency of computation, in which cases the methods could easily be extended to weaker versions of these conditions.

- Sites evolve independently.

Our method calculates the derivatives of the likelihood of a given site. In order to extend this to obtain the overall likelihood of the data, we need to have a method of obtaining the overall likelihood from the likelihood at each site. Independence allows us to calculate the overall log-likelihood as the sum of the log-likelihoods at each site. We could relax this condition to the assumption that sites evolve independently conditional on certain parameters, which might have certain correlations. (Examples of such models are hidden Markov models, see Yang (1995) and Felsenstein and Churchill (1996).)

- Different branches of the tree evolve independently.

This assumption is important for similar reasons to the previous assumption, and as in the previous assumption, it would be possible to extend the methods described in this paper to deal with the cases where there is some dependence between parameters on different branches but conditional independence of the evolution on different branches holds (e.g. some covarion models).

- Stationarity.

This is convenient because it allows us reroot the tree and therefore reduce the total calculation by reusing the same calculations. (Assuming a relaxed form of reversibility — see Reversibility for more details).

- Reversibility.

This is important for two reasons. Firstly, it guarantees that the Q-matrix is diagonalisable over the real numbers, and allows us to use special methods for diagonalising the Q-matrix, which do not apply to general matrices. Without this assumption, calculating e^{Qt} and its derivatives becomes more complicated. Secondly,

it means that we do not need to worry about the directions of branches, which simplifies the process of rerooting the tree. If we are careful about the directions of each branch, we can still reroot the tree without the reversibility assumption, but stationarity might be violated if the largest left and right eigenvectors of the Q-matrix are not equal.

- Homotachy.

This refers to the assumption that the rates of substitution are the same across all lineages for a given site. Since our method is based upon a model where the parameters are different on different branches, it trivially extends to deal with heterotachy (provided other assumptions are not violated).

2.2 Differentiating the P Matrix

In order to calculate any of the derivatives, we need to be able to differentiate the P-matrix with respect to the parameters. With respect to the branch length, this is easy. Recall $P = e^{Qt}$, so $\frac{\partial P}{dt} = Qe^{Qt}$ and $\frac{\partial^2 P}{dt^2} = Q^2 e^{Qt}$. However, differentiating with respect to other parameters is more difficult, because matrix multiplication doesn't commute. This is of course a well-known problem, and there are a number of algorithms for computing the derivative of a matrix exponential with respect to parameters which affect the matrix. The method we use was applied to linear differential equations by Jennrich and Bright (1976), and to continuous time Markov processes by Kalbfleisch and Lawless (1985). However, they elected to use a quasi-Newton method, rather than calculate the second derivative of log-likelihood with respect to the parameters. We did not find calculations of second derivatives in the literature. Thus we will provide the results in the following theorem and give a detailed derivation in Appendix 1.

Theorem 1. *Suppose parameters β and γ are two parameters which influence the Q matrix, and the following derivatives are readily available: $\frac{\partial Q}{\partial \beta} = M_\beta$, $\frac{\partial Q}{\partial \gamma} = M_\gamma$ and $\frac{\partial^2 Q}{\partial \beta \partial \gamma} = M_{\beta\gamma}$. Then we have:*

1. *If the process is reversible, then for any parameter values, Q is diagonalisable as $Q = ADA^{-1}$.*

2. *For any invertible constant matrix C, we define $X = C^{-1}QC$, so that $Q = CXC^{-1}$, and we have that*

$$\frac{\partial P}{\partial \beta} = \frac{\partial}{\partial \beta}\left(e^{Qt}\right) = C\frac{\partial (e^{Xt})}{\partial \beta}C^{-1},$$

and we define $N_\beta = \frac{\partial X}{\partial \beta} = C^{-1} M_\beta C$. *Now, for a given set of parameter values* θ_0, *if* $Q(\theta_0)$ *is diagonalizable as* $Q(\theta_0) = ADA^{-1}$, *then by choosing* $C = A$ *in the above expression,* $X(\theta_0)$ *is equal to the diagonal matrix D with entries* d_i, *and the* ij^{th} *entry of the matrix* $\left.\frac{\partial (e^{Xt})}{\partial \beta}\right|_{\theta_0}$ *is given by*

$$\left(\left.\frac{\partial (e^{Xt})}{\partial \beta}\right|_{\theta_0} \right)_{ij} = \begin{cases} \frac{(N_\beta)_{ij}(e^{d_i t} - e^{d_j t})}{d_i - d_j} & \text{if } d_i \neq d_j \\ (N_\beta)_{ij} t e^{d_i t} & \text{if } d_i = d_j \end{cases}$$

Note that the above result for $d_i = d_j$ *is also the limit of the fractional form for* $d_i \neq d_j$, *when* $d_j \to d_i$, *as can be seen e.g. by using L'Hopîtal's rule.*

3. *Furthermore, denoting* $N_\gamma = A^{-1} M_\gamma A$ *and* $N_{\beta\gamma} = A^{-1} M_{\beta\gamma} A$, *the second derivative of the transition matrix P with respect to parameters* β *and* γ *(which could be the same parameter) can be written as*

$$\left.\frac{\partial^2 P}{\partial \beta \partial \gamma}\right|_{\theta_0} = A \left.\frac{\partial^2 (e^{Xt})}{\partial \beta \partial \gamma}\right|_{\theta_0} A^{-1},$$

where the ij^{th} *entry of matrix* $\left.\frac{\partial^2 (e^{Xt})}{\partial \beta \partial \gamma}\right|_{\theta_0}$ *is given by the following, with appropriate limiting values taken in the cases when* $d_i = d_j$, $d_i = d_k$, *or* $d_j = d_k$:

$$\left(\left.\frac{\partial^2 (e^{Xt})}{\partial \beta \partial \gamma}\right|_{\theta_0} \right)_{ij} = \frac{(N_{\beta\gamma})_{ij}(e^{d_i t} - e^{d_j t})}{d_i - d_j} + \sum_k \left((N_\beta)_{ik}(N_\gamma)_{kj} + (N_\gamma)_{ik}(N_\beta)_{kj} \right) h_{ijk}$$

where $h_{ijk} = \frac{e^{d_i t}}{(d_i - d_j)(d_i - d_k)} + \frac{e^{d_j t}}{(d_j - d_i)(d_j - d_k)} + \frac{e^{d_k t}}{(d_k - d_j)(d_k - d_i)}$.

2.3 Recap of Pruning Algorithm

The pruning algorithm, introduced in Felsenstein (1973) and applied more efficiently to unrooted trees in Felsenstein (1981), is a method for efficiently computing the likelihood of the data for one site, given a fixed tree with a root selected and instantaneous substitution matrix Q. The transition matrix for a branch of length t is therefore given by $P(t) = e^{Qt}$. The basic idea of the pruning algorithm is that for a subtree, the only information that affects the overall likelihood is the list of conditional likelihoods, conditional on the value at the root of the subtree. Once this observation is made, the lists of conditional likelihoods can be built up starting with the leaves, until the root is reached, at which point the likelihood can be directly computed.

We will denote the likelihood list of the subtree below node N, conditional on the value at N, as a vector d_N and call it the Down1 list at N. If N is not the root

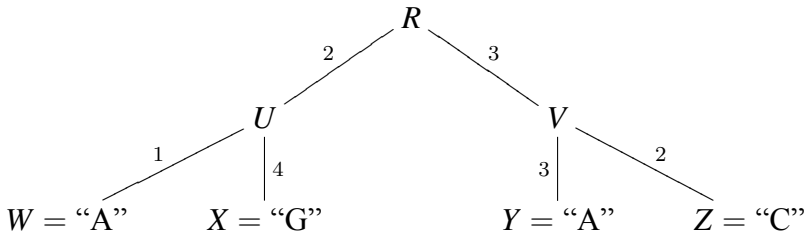


Figure 1: Example Tree. Numbers on branches are branch lengths (expected number of substitutions per site).

of the tree, and the node directly above N is M with an edge e between N and M, then $\delta_N = e^{Qt_e} d_N$ will be called the Down2 list at N. For convenience we use # to denote the elementwise product of two vectors of the same length. The list d_N can be recursively formed as $d_N = \delta_{N_1} \# \delta_{N_2} \# \cdots \# \delta_{N_k}$, where $N_1, \ldots, N_k$ are the immediate descendants of node N along branches of lengths $t_1, \ldots, t_k$ respectively, and $\delta_{N_i} = e^{Qt_i} d_{N_i}$.

2.3.1 Illustrative Example: Pruning Algorithm

We illustrate the pruning algorithm by working through an example tree (Figure 1). To simplify the example we use nucleotide data, but the same method can be applied to amino acid or codon data. We will later use the same tree to demonstrate our algorithm for calculating the derivatives.
Suppose that the Q matrix is such that for branch length equal to 1,

$$P(1) = e^{Q} = \begin{array}{c} \begin{array}{cccc} \text{A} & \text{C} & \text{G} & \text{T} \end{array} \\ \left(\begin{array}{cccc} 0.3 & 0.2 & 0.4 & 0.1 \\ 0.2 & 0.1 & 0.4 & 0.3 \\ 0.4 & 0.4 & 0.1 & 0.1 \\ 0.1 & 0.3 & 0.1 & 0.5 \end{array} \right) \begin{array}{c} \text{A} \\ \text{C} \\ \text{G} \\ \text{T} \end{array} \end{array}$$

This allows us to calculate the cases for branch length equal to 2, 3 and 4 respectively:

$$P(2) = e^{2Q} = \begin{pmatrix} 0.30 & 0.27 & 0.25 & 0.18 \\ 0.27 & 0.30 & 0.19 & 0.24 \\ 0.25 & 0.19 & 0.34 & 0.22 \\ 0.18 & 0.24 & 0.22 & 0.36 \end{pmatrix}$$

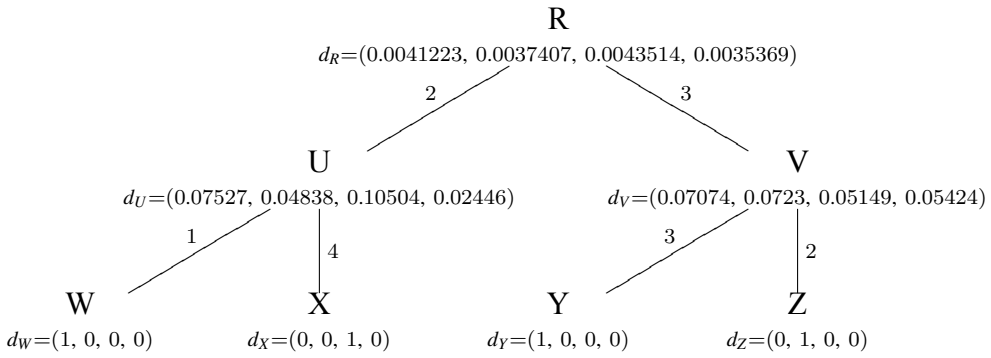


Figure 2: Likelihood Lists for Example Tree

$$P(3) = e^{3Q} = \begin{pmatrix} 0.262 & 0.241 & 0.271 & 0.226 \\ 0.241 & 0.232 & 0.271 & 0.256 \\ 0.271 & 0.271 & 0.232 & 0.226 \\ 0.226 & 0.256 & 0.226 & 0.292 \end{pmatrix}$$

$$P(4) = e^{4Q} = \begin{pmatrix} 0.2578 & 0.2527 & 0.2509 & 0.2386 \\ 0.2527 & 0.2566 & 0.2419 & 0.2488 \\ 0.2509 & 0.2419 & 0.2626 & 0.2446 \\ 0.2386 & 0.2488 & 0.2446 & 0.2680 \end{pmatrix}$$

For nucleotide data, the likelihood list d_N at node N is a 4-dimensional vector. We give our lists in alphabetical order, so the first element is the likelihood conditional on the nucleotide at that node being "A", denoted as $d_{N,\mathrm{A}}$, etc. Thus $d_N = (d_{N,\mathrm{A}}, d_{N,\mathrm{C}}, d_{N,\mathrm{G}}, d_{N,\mathrm{T}})$. We start by forming the likelihood lists at the leaf nodes, W, X, Y and Z. Since we know what the nucleotide is at the leaf nodes, the likelihoods in the list are all either 1 or 0 — 1 for the nucleotide at that node, and 0 otherwise, so for example $d_W = (1, 0, 0, 0)$, because the nucleotide at W is "A".

Next we want to form the likelihood lists at the parent nodes of leaf nodes, which in this case are d_U and d_V. To form the list d_U, we see that the likelihood of the subtree below U, conditional on the nucleotide at U, is given by the product of the two conditional likelihoods δ_W and δ_X. For example, conditional on the nucleotide at U being "A", the likelihood of "A" at W is $(e^Q)_{11}$, which is 0.3. Similarly the conditional likelihood of "G" at X is $(e^{4Q})_{13}$, which is 0.2509. Therefore, the overall likelihood of the tree below U, conditional on the nucleotide at U being "A" is $0.3 \times 0.2509 = 0.07527$. We can use this method to calculate the conditional likelihood lists at U and V, d_U and d_V, and then form $\delta_U = e^{2Q} d_U$ and $\delta_V = e^{3Q} d_V$. The conditional likelihood list at R is $d_R = \delta_U \# \delta_V$. This gives the likelihood lists in Figure 2.

From the top likelihood list, we can read off the overall likelihood by summing the nucleotide frequencies (which we will henceforth refer to as π) multiplied by the corresponding conditional likelihoods. That is, the overall likelihood is given by $\sum_{i=1}^{4} d_{R,i}\pi_i = \pi^T d_R$, where $(\_)^T$ denotes transposition. In this example, the nucleotide frequencies are all $\frac{1}{4}$, (the vector of these frequencies must be the eigenvector of e^Q with eigenvalue 1) thus, the overall likelihood is 0.0039378.

2.4 First Derivatives

Our method for calculating the first derivatives with respect to a parameter on a given branch is the same as Schadt et al (1998), but we will see in later sections that when we are computing the second derivatives as well, we can calculate the first derivatives more efficiently.

Suppose we want to calculate the derivative of the site likelihood with respect to a parameter β on some branch e of the tree, of length t_e. Denote the root of the tree by R, we then have

$$\frac{\partial L}{\partial \beta_e} = \frac{\partial}{\partial \beta_e}\left(\pi^T d_R\right) = \pi^T \frac{\partial d_R}{\partial \beta_e}$$

The likelihood list d_R was computed recursively using the pruning algorithm, thus we can calculate its derivative by applying the following formulae recursively. For any node N with the nodes $N_1, \ldots, N_k$ immediately below it ($k = 0$ if N is a leaf node, $k = 2$ if N is not a leaf node and the tree is bifurcating), we use the formulae

$$\frac{\partial d_N}{\partial \beta_e} = \begin{cases} \delta_{N_1} \# \cdots \# \frac{\partial \delta_{N_j}}{\partial \beta_e} \# \cdots \# \delta_{N_k} & \text{if } e \text{ is somewhere below} N_j \text{ or} \\ & e \text{ is the edge between } N \text{ and } N_j \\ 0 & \text{if } e \text{ is not below } N \end{cases}$$

$$\frac{\partial \delta_{N_j}}{\partial \beta_e} = \begin{cases} \frac{\partial e^{Qt_e}}{\partial \beta_e} d_{N_j} & \text{if } e \text{ is the edge between } N \text{ and } N_j \\ e^{Qt_e} \frac{\partial d_{N_j}}{\partial \beta_e} & \text{if } e \text{ is somewhere below} N_j \end{cases}$$

To explain the first formula, Recall that $d_N = \delta_{N_1} \# \ldots \# \delta_{N_k}$. Therefore, by the product rule, we have that $\frac{\partial d_N}{\partial \beta_e} = \sum_{j=1}^{k} \delta_{N_1} \# \cdots \# \frac{\partial \delta_{N_j}}{\partial \beta_e} \# \cdots \# \delta_{N_k}$. However, if e is not below N_j, or between N_j and N, then δ_{N_j} does not depend on β_e, so $\frac{\partial \delta_{N_j}}{\partial \beta_e} = 0$. Therefore at most one term in the sum is non-zero.

In effect, this is just the same calculation as computing the likelihood using the pruning algorithm, except that on the branch e, the matrix e^{Qt_e} has been replaced by its derivative.

Recall that for a time-reversible model, the tree can be rerooted before applying the pruning algorithm. This is also true for our derivative calculation. For the derivative $\frac{\partial L}{\partial \beta_e}$, it will often be convenient to reroot the tree at the top node of e. Suppose the bottom node of e is N and the top node of e is M, the likelihood can be rewritten as $d_N^T(e^{Qt_e})^T \Pi u_N$, where Π is a diagonal matrix whose diagonal entries are π, and u_N, called the Up1 list at N, is the likelihood list conditional on the values at M for the tree obtained by cutting the branch e, and rerooting at M. The list u_N can be obtained as a # product of Down2 lists for all the nodes below M in the new subtree, or equivalently, it is the Down1 list d_M when we reroot the original tree at N. Thus we now have $\frac{\partial L}{\partial \beta_e} = d_N^T(\frac{\partial e^{Qt_e}}{\partial \beta_e})^T \Pi u_N$.

By properly rerooting the tree, we can reuse the likelihood lists and speed up computation of the first and second derivatives. A full description of all the lists necessary is given in Section 3.

2.4.1 Illustrative Example: First Derivatives

In the example above, we had calculated the likelihood lists d_U, d_V and d_R. Suppose we want to calculate the derivative of the likelihood with respect to a parameter β on the branch RU. Note that the site likelihood is given by $\pi^T(\delta_U \# \delta_V)$, or equivalently $(d_U)^T(e^{2Q})^T \Pi \delta_V$. Thus the derivative of the site likelihood with respect to β on RU is $(d_U)^T \left(\frac{\partial e^{2Q}}{\partial \beta_{RU}}\right)^T \Pi \delta_V$.

For a different branch, for example WU, we can reroot the tree so that U becomes the root. Now we can write the likelihood as $L = (d_w)^T(e^Q)^T \Pi u_W$. Thus the derivative of the site likelihood with respect to β on WU can be written as $(d_W)^T \left(\frac{\partial e^Q}{\partial \beta_{WU}}\right)^T \Pi u_W$.

2.5 Second Derivatives

The second derivative of the likelihood is calculated as $\sum_e \sum_{e'} \frac{\partial^2 L}{\partial \beta_e \partial \gamma_{e'}}$ for all pairs of branches e and e'. We divide this sum into two cases:

1. $e = e'$: terms arising from taking the second derivative of the likelihood with respect to branch parameters on the same branch, $\sum_e \frac{\partial^2 L}{\partial \beta_e \partial \gamma_e}$.

In this case, the calculation is similar to calculating a first derivative, except that instead of replacing the P-matrix by its first derivative, we replace it by its second derivative.

2. $e \neq e'$: terms arising from taking the second derivative of the likelihood with respect to branch parameters on different branches, $\sum_{e \neq e'} \frac{\partial^2 L}{\partial \beta_e \partial \gamma_{e'}}$.

We will always reroot the tree so that one of the following two situations applies:

- The root is at one end, denoted M, of edge e', and the other edge e is below e'. The relative position of e and e' is denoted $e < e'$. Suppose the other end of e' is node N. Then the likelihood can be written as $L = (d_N)^T (e^{Qt_{e'}})^T \Pi u_N$. The second derivative is given by $\frac{\partial^2 L}{\partial \beta_e \partial \gamma_{e'}} = \left(\frac{\partial d_N}{\partial \beta_e}\right)^T \left(\frac{\partial e^{Qt_{e'}}}{\partial \gamma_{e'}}\right)^T \Pi u_N$.
- The root is at a node N between e and e'. The relative position of e and e' is now denoted as $e//e'$. Let the direct descendants of the node N be $N_1, \ldots, N_k$, then the likelihood is $L = \pi^T(\delta_{N_1} \# \delta_{N_2} \# \cdots \# \delta_{N_k})$. Suppose e is below N_1, or is the edge between N and N_1 (henceforth, we denote this situation $e \leqslant N_1$) and similarly $e' \leqslant N_2$. Then the derivative is $\frac{\partial^2 L}{\partial \beta_e \partial \gamma_{e'}} = \pi^T \left(\left(\frac{\partial \delta_{N_1}}{\partial \beta_e}\right) \# \left(\frac{\partial \delta_{N_2}}{\partial \gamma_{e'}}\right) \# \cdots \# \delta_{N_k}\right)$.

Obviously for these terms, differentiating the likelihood requires us to differentiate the lists δ_N and d_N for various nodes N. Since these differentiated lists need to be calculated recursively, we want to choose the root which will minimise the number of lists we need to compute. How to achieve the best efficiency in calculating these terms will be given in Section 3.

2.5.1 Illustrative Example: Calculating the Second Derivatives for Parameters on Different Branches

We demonstrate how to calculate the second derivatives with respect to parameters on two different branches for each of the cases $e < e'$ and $e//e'$.

- $e = UW, e' = RU$: this is the case $e < e'$. The likelihood of the tree, when rooted at R, is given by $L = d_U^T (e^{2Q})^T \Pi u_U$. Therefore, we get $\frac{\partial^2 L}{\partial \beta_{UW} \partial \gamma_{RU}} = \left(\frac{\partial d_U}{\partial \beta_{UW}}\right)^T \left(\frac{\partial e^{2Q}}{\partial \gamma_{RU}}\right)^T \Pi u_U$, where $\frac{\partial d_U}{\partial \beta_{UW}} = \left(\frac{\partial e^Q}{\partial \beta_{UW}} d_W\right) \# \delta_X$.
- $e = UW, e' = RV$: this is the case $e//e'$. Again we root the tree at R, and the likelihood can be expressed as $L = \pi^T(\delta_U \# \delta_V)$. Now we see that $\frac{\partial^2 L}{\partial \beta_{UW} \partial \gamma_{RV}} = \pi^T \left(\left(\frac{\partial \delta_U}{\partial \beta_{UW}}\right) \# \left(\frac{\partial \delta_V}{\partial \gamma_{RV}}\right)\right)$. We can calculate the derivatives of the two lists recursively: $\frac{\partial \delta_U}{\partial \beta_{UW}} = e^{2Q} \frac{\partial d_U}{\partial \beta_{UW}}$ and $\frac{\partial \delta_V}{\partial \gamma_{RV}} = \frac{\partial e^{3Q}}{\partial \gamma_{RV}} d_V$.

2.6 Mixed Models

In phylogenetic analysis, rates-across-sites variation (and across lineages) is now common practice. Since our calculations are derived from a model with heterotachy, they can easily be extended to allow rate variation acros lineages. We now extend

our calculations to deal with cases where some parameters can be assumed to have a random effect across sites, i.e. their values at a given site follow a probability distribution. A particularly important example is the nonsynonymous/synonymous ratio in the codon models (Goldman and Yang, 1994). Yang (1994), Nielsen & Yang (1998), Yang, Nielsen, Goldman & Pedersen (2000) propose a number of different models for how this parameter should be allowed to vary among sites. These models are frequently used to detect positive selection. Our methods here can also be used to calculate the derivatives with respect to parameters in those models, and therefore make it possible to implement Newton-Raphson methods to maximise likelihood or perform inference based on likelihood theory, for their models.

Assuming a vector β of parameters which affect the Q matrix has random effects across sites, the likelihood for a given site is now given by

$$\int L_\beta P(\beta) d\beta$$

where L_β is the likelihood given the values of β and $P(\beta)$ is the probability density function of β. For simplicity, and to make the computation feasible, we will consider a discrete model for $P(\beta)$, where there are a finite number of possible values for β, which we denote $\beta_1, \beta_2, \ldots, \beta_k$, with probabilities $p_1, p_2, \ldots, p_k$ respectively. This approximation is common practice. The likelihood for a given site can now be written as

$$L = \sum_{i=1}^{k} p_i L_i$$

where L_i is the likelihood when $\beta = \beta_i$. Now it is easy to see that the derivative with respect to β_i is just $p_i \frac{\partial L_i}{\partial \beta_i}$, and the derivative of the log likelihood is $\frac{p_i}{L_i} \frac{\partial L_i}{\partial \beta_i}$. Second derivatives can also be worked out easily. For derivatives with respect to the probabilities p_i, it is easy to see that $\frac{\partial L}{\partial p_i} = L_i$.

However since the p_i are constrained to sum to 1, we need to find a suitable parametrisation of these probabilities. One way to parametrise a set of probabilities is to set values $\phi_i = \frac{p_i}{p_k}$ for $i < k$. This means that $p_k = \frac{1}{1+\sum_{i=1}^{k-1} \phi_i}$, and $p_i = \phi_i p_k$. Now to find the derivatives of likelihood with respect to ϕ_i, we merely need to find the derivatives with respect to the p_i, and use the formulae $\frac{\partial p_k}{\partial \phi_j} = -{p_k}^2$, and

$$\frac{\partial p_i}{\partial \phi_j} = \begin{cases} p_k - p_i p_k & \text{if } i = j \\ -p_i p_k & \text{otherwise} \end{cases}$$

We can use the product rule to calculate second derivatives. Using these, we can work out the full Hessian matrix with respect to all the parameters. In terms of computation, we have to calculate the Hessian for each possible parameter value in the

mixture, so the complexity is proportional to the number of categories in the distribution $P(\beta)$. We will therefore want to keep this number fairly small. When $P(\beta)$ is a continuous distribution, it is usually possible to choose a discrete approximation so that the number of different parameter values is small, but the accuracy of the approximation is still sufficient for the purposes.

3 Implementation

In this section we extend the theory to obtain an efficient implementation that could be used to write a program for calculating the first and second derivatives of likelihood. We begin by describing full sets of lists that are necessary for the efficient implementation, followed by two sections that deal with the computational issues in differentiating the P matrix twice. We then present the full algorithm in detail. Then in Section 3.5, we analyse the time complexity of the algorithm, and show that it is sufficiently efficient to be used in practice for computing first and second derivatives of likelihood.

3.1 Different Types of Lists

The calculation of second derivatives as described above is very slow. Many of the calculations involved are repeated multiple times. In this section, we describe how to better organise the computations, in order to reduce the amount of needless repetition.

There are two main sources of repeated computation. Firstly, when we calculate the derivative of likelihood with respect to a branch parameter β_e (which could be branch length), we have obtained the derivative of the P matrix in the form AXA^{-1}, for some matrix X (see Theorem 1). Therefore, we are computing the derivative as $\frac{\partial L}{\partial \beta_e} = d_N^T (AXA^{-1})^T \Pi u_N = u_N^T \Pi AXA^{-1} d_N$. The most efficient way to calculate this is $({u_N}^T \Pi A) X (A^{-1} d_N)$, so by saving the results for ${u_N}^T \Pi A$ and $A^{-1} d_N$ and reusing them to avoid repeatedly calculating them for every branch parameter, we can reduce the computation time.

Secondly, for Q-matrix parameters β and γ, we calculate the derivative $\frac{\partial^2 L}{\partial \beta \partial \gamma}$ as the sum $\sum_{e,e'} \frac{\partial^2 L}{\partial \beta_e \partial \gamma_{e'}}$, where the exact form of the summands depends on the relative positions of e and e' in the tree. When the root is at a node N between e and e', for example, the summand is given by $\frac{\partial^2 L}{\partial \beta_e \partial \gamma_{e'}} = \pi^T \left(\left(\frac{\partial \delta_{N_1}}{\partial \beta_e} \right) \# \left(\frac{\partial \delta_{N_2}}{\partial \gamma_{e'}} \right) \# \cdots \# \delta_{N_k} \right)$, where $N_1, \ldots, N_k$ are the immediate descendants of N and $e \leqslant N_1$, $e' \leqslant N_2$. If we collect

similar terms into the following sum and factorise it as:

$$\sum_{\substack{e\leqslant N_1\\ e'\leqslant N_2}} \pi^T\left(\left(\frac{\partial\delta_{N_1}}{\partial\beta_e}\right)\#\left(\frac{\partial\delta_{N_2}}{\partial\gamma_{e'}}\right)\#\cdots\#\delta_{N_k}\right)=\pi^T\left(\left(\sum_{e\leqslant N_1}\frac{\partial\delta_{N_1}}{\partial\beta_e}\right)\#\left(\sum_{e'\leqslant N_2}\frac{\partial\delta_{N_2}}{\partial\gamma_{e'}}\right)\#\cdots\#\delta_{N_k}\right)$$

we can massively reduce the computation by computing the lists such as $\gamma^{\beta}_{N_1} = \sum_{e\leqslant N_1}\frac{\partial\delta_{N_1}}{\partial\beta_e}$.

We now provide a summary of all kinds of lists that need to be compiled, along with their definitions at a node N for a rooted tree. The recursive formulae for calculating these lists can easily be obtained from the basic lists d_N, δ_N and u_N. The recursive formulae are presented explicitly in Section 3.4. For convenience, we denote the node immediately above N (the parent of N) as M if N is not the root of the tree.

Down1 This is the basic list d_N that we used in the pruning algorithm. The values are the conditional likelihoods of the tree below node N. We will also be interested in the derivatives of this list with respect to a particular branch-length parameter. We will use the notation $d^{t_e}_N = \frac{\partial d_N}{\partial t_e}$.

Down2 This is the list δ_N in the *Theory* section. It is the list of conditional likelihoods at the parent, M, of N, for the tree consisting of M, N, and all nodes below N. It is given by $\delta_N = e^{Qt}d_N$. Again, we denote $\delta^{t_e}_N = \frac{\partial\delta_N}{\partial t_e}$ for a branch e.

Down3 This is the list given as $s_N = A^{-1}d_N$.

Schadt, Sinsheimer and Lange (1998) refer to the list δ_N as either R_M or L_M, where M is the parent of N, and R or L is used to indicate that N is reached from the right or left branch of M respectively. For multifurcating trees, this notation cannot be easily applied, so we prefer to associate this list to the lower node. This means that the number of lists that we associate with each node is fixed, thereby simplifying the process of coding the algorithm.

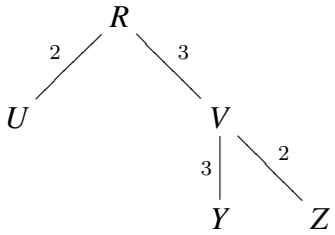
Up1 This is the list u_N that we used for calculating the first derivatives. If we let M be the parent of N, then the list u_N is the Down1 list, for the tree obtained by rerooting at N, at the node M. We also want to calculate the list $u^{t_e}_N = \frac{\partial u_N}{\partial t_e}$ for all edges e that are below M after we reroot the tree at N.

Note that for our example tree, the root has only two branches, thus we have $\delta_U = u_V$. If the root had more branches, then u_V would include δ_U and other extra branches.

Up2 This list, denoted as v_N, bears a similar relation to the list u_N to the relation that δ_N bears to d_N. It corresponds to the conditional likelihoods of the subtree obtained by removing all nodes below N. It is given by $v_N = e^{Qt}u_N = \Pi^{-1}(e^{Qt})^T\Pi u_N$. We also want to calculate the list $v_N^{t_e} = \frac{\partial v_N}{\partial t_e}$ for all edges e not below N ($e \geqslant N$).

Up3 This is the list given as $w_N = A^T\Pi u_N$. We also want to calculate the list $w_N^{t_e} = \frac{\partial w_N}{\partial t_e} = A^T\Pi u_N^{t_e}$ for all edges $e > N$.

For example, the list v_U for the example tree corresponds to the subtree



conditional on the nucleotide at U.

Cumulative1 This is the cumulative list $c_N^\beta = \sum_{e<N} \frac{\partial(d_N)}{\partial\beta_e} = \frac{\partial(d_N)}{\partial\beta}$, where the sum is taken over all branches below N in the tree.

Cumulative2 As we noted above, the lists δ_N are useful for making a more efficient computation. We therefore want a similar sort of cumulative list, $\gamma_N^\beta = \sum_{e\leqslant N} \frac{\partial(\delta_N)}{\partial\beta_e} = \frac{\partial(\delta_N)}{\partial\beta}$, where the sum is taken over all branches below N in the tree, and also the branch directly above N.

A summary of all the above defined lists and their notations is given in Table 1.

Table 1: Lists and their notations

list	Down1	Up1	Down2	Up2	Down3	Up3	Cumu-1	Cumu-2
notation	d_N	u_N	δ_N	v_N	s_N	w_N	c_N	γ_N

We now work through the use of the cumulative lists in the example tree to show how the cumulative lists can reduce the amount of computation in that case. This is summarised in Table 2.

For two Q-matrix parameters β and γ, Recall that the derivative $\frac{\partial^2 L}{\partial\beta\partial\gamma}$ is given as $\sum_{e,e'} \frac{\partial^2 L}{\partial\beta_e\partial\gamma_{e'}}$, and that furthermore, we divided this sum into four sums.

$$\frac{\partial^2 L}{\partial\beta\partial\gamma} = \sum_{e<e'} \frac{\partial^2 L}{\partial\beta_e\partial\gamma_{e'}} + \sum_{e>e'} \frac{\partial^2 L}{\partial\beta_e\partial\gamma_{e'}} + \sum_{e//e'} \frac{\partial^2 L}{\partial\beta_e\partial\gamma_{e'}} + \sum_{e=e'} \frac{\partial^2 L}{\partial\beta_e\partial\gamma_{e'}}$$

Using the cumulative lists, the above sums can be rewritten as sums over the nodes on the example tree, as follows:

$$\sum_{N=U,V} {u_N}^T \Pi \frac{\partial P(t_{e'})}{\partial \gamma_{e'}} c_N^\beta + \sum_{N=U,V} {u_N}^T \Pi \frac{\partial P(t_e)}{\partial \beta_e} c_N^\gamma +$$
$$\pi^T \sum_{N=R,U,V} \left(v_N \# \gamma_{N_1}^\beta \# \gamma_{N_2}^\gamma + v_N \# \gamma_{N_1}^\gamma \# \gamma_{N_2}^\beta \right) + \sum_e \frac{\partial^2 L}{\partial \beta_e \partial \gamma_e}$$

This reduces the original sums over 30 pairs of distinct edges $e \neq e'$ to just 10 terms. as shown in Table 2. The terms with $e = e'$ are dealt with in the next section.

Table 2: Illustrative example: cumulative lists cover components of the sum $\sum_{e \neq e'} \frac{\partial^2 L}{\partial \beta_e \partial \gamma_{e'}}$ for distinct pairs of branches, with 30 pairs of branches reduced to just 10 terms.

		e'					
		RU	UW	UX	RV	VY	VZ
e	RU		l_1				
	UW	l_3		l_7	l_5		
	UX		l_8				
	RV					l_2	
	VY	l_6			l_4	l_9	
	VZ					l_{10}	

$$l_1 = {u_U}^T \Pi \frac{\partial P(t_{RU})}{\partial \beta} c_U^\gamma \qquad l_2 = {u_V}^T \Pi \frac{\partial P(t_{RV})}{\partial \beta} c_V^\gamma \qquad l_3 = {u_U}^T \Pi \frac{\partial P(t_{RU})}{\partial \gamma} c_U^\beta$$
$$l_4 = {u_V}^T \Pi \frac{\partial P(t_{RV})}{\partial \gamma} c_V^\beta \qquad l_5 = \pi^T (\gamma_U^\beta \# \gamma_V^\gamma) \qquad l_6 = \pi^T (\gamma_U^\gamma \# \gamma_V^\beta)$$
$$l_7 = \pi^T (v_U \# \gamma_W^\beta \# \gamma_X^\gamma) \qquad l_8 = \pi^T (v_U \# \gamma_X^\beta \# \gamma_W^\gamma) \qquad l_9 = \pi^T (v_V \# \gamma_Y^\beta \# \gamma_Z^\gamma)$$
$$l_{10} = \pi^T (v_V \# \gamma_Y^\gamma \# \gamma_Z^\beta)$$

3.2 Speeding up Computation of Second Derivatives of the *P*-Matrix

Calculating the second derivative of e^{Dt} with respect to two parameters in the way described in Theorem 1 requires a lot of computation. Recall that:

$$\left(\frac{\partial^2(e^{Dt})}{\partial\beta\partial\gamma}\right)_{ij} = \frac{(N_{\beta\gamma})_{ij}(e^{d_i t} - e^{d_j t})}{d_i - d_j} + \sum_k \left((N_\beta)_{ik}(N_\gamma)_{kj} + (N_\gamma)_{ik}(N_\beta)_{kj}\right) h_{ijk}$$

where $h_{ijk} = \frac{e^{d_i t}}{(d_i - d_j)(d_i - d_k)} + \frac{e^{d_j t}}{(d_j - d_i)(d_j - d_k)} + \frac{e^{d_k t}}{(d_k - d_j)(d_k - d_i)}$.
Calculating this takes $O(n^3)$ operations, where n is the number of rows (or columns) of the Q-matrix (so 4 for nucleotide data, 20 for amino acid data and 61 for codon data in standard genetic code), and it needs to be calculated for each branch, and for each pair of parameters, and each site. This leads to complexity of $O(bp^2n^3S)$, where b is the number of branches, p is the number of parameters and S is the number of sites.

These second derivatives are used to calculate terms of the form $\sum_e \frac{\partial^2 L}{\partial\beta_e\partial\gamma_e}$, which are given by the formula

$$\sum_e \frac{\partial^2 L}{\partial\beta_e\partial\gamma_e} = \sum_e w_N^T \frac{\partial^2 e^{Dt_e}}{\partial\beta_e\partial\gamma_e} s_N$$

where N is the bottom node of e. If we ignore for the moment the cases where two of d_i, d_j and d_k have equal values, we can break the second terms for each edge in the above sum into three separate sums:

$$\text{Sum 1} = \sum_k \left((N_\beta)_{ik}(N_\gamma)_{kj} + (N_\gamma)_{ik}(N_\beta)_{kj}\right) \frac{e^{d_i t}}{(d_i - d_j)(d_i - d_k)}$$

$$\text{Sum 2} = \sum_k \left((N_\beta)_{ik}(N_\gamma)_{kj} + (N_\gamma)_{ik}(N_\beta)_{kj}\right) \frac{e^{d_j t}}{(d_j - d_i)(d_j - d_k)}$$

$$\text{Sum 3} = \sum_k \left((N_\beta)_{ik}(N_\gamma)_{kj} + (N_\gamma)_{ik}(N_\beta)_{kj}\right) \frac{e^{d_k t}}{(d_k - d_j)(d_k - d_i)}$$

The benefit of doing this is that for Sum 1 and Sum 2, the parts that depend on the branch length are constant factors, so the sum can be performed once for all branches, and for all sites. The part depending on the branch lengths does not depend on the parameters β and γ. Thus we can compute the term in $\sum_e \frac{\partial^2 L}{\partial\beta_e\partial\gamma_e}$ that is derived from Sum 1 as

$$\sum_{i,j}\left(\sum_N w_{N,i} s_{N,j} e^{d_i t}\right)\left(\sum_k \frac{\left((N_\beta)_{ik}(N_\gamma)_{kj} + (N_\gamma)_{ik}(N_\beta)_{kj}\right)}{(d_i - d_j)(d_i - d_k)}\right)$$

Thus for each site, it requires $O(bn^2)$ operations to compute the first term (sum over all branches), and $O(p^2n^2)$ operations to calculate these sums for each pair of parameters.

We can rewrite Sum 3 as:

$$\sum_k \frac{(N_\beta)_{ik}}{(d_k - d_i)} \times \frac{(N_\gamma)_{kj}}{(d_k - d_j)} \times e^{d_k t} + \sum_k \frac{(N_\gamma)_{ik}}{(d_k - d_i)} \times \frac{(N_\beta)_{kj}}{(d_k - d_j)} \times e^{d_k t}$$

These are just matrix products, and so, when we want to calculate $w_N^T \frac{\partial^2 e^{Dte}}{\partial \beta_e \partial \gamma_e} s_N$, the formula is

$$\sum_{i,j,k} w_{N,i} s_{N,j} \frac{(N_\beta)_{ik}}{(d_k - d_i)} \times \frac{(N_\gamma)_{kj}}{(d_k - d_j)} \times e^{d_k t} + \sum_{i,j,k} w_{N,i} s_{N,j} \frac{(N_\gamma)_{ik}}{(d_k - d_i)} \times \frac{(N_\beta)_{kj}}{(d_k - d_j)} \times e^{d_k t}$$

We can break the first term up as

$$\sum_k \left(\sum_i w_{N,i} \frac{(N_\beta)_{ik}}{(d_k - d_i)} \right) \times \left(\sum_j s_{N,j} \frac{(N_\gamma)_{kj}}{(d_k - d_j)} \right) \times e^{d_k t}$$

and the second term can be broken up similarly.

Now each of the inner sums can be performed just once for each parameter, requiring a total of $O(bpn^2S)$ operations. In fact, the sum $t_N^\beta = \sum_i w_{N,i} \frac{(N_\beta)_{ik}}{(d_k - d_i)}$ is part of the computation of the cumulative list γ_N^β, so we can simply store the values when we calculate them at that time. The outer sum requires only $O(bp^2n)$ operations at each site. This means that the overall calculation of this component of the second derivative requires only $O(bpn^2 + bp^2n)$ operations for each site, and in preparation, we need $O(p^2n^3)$ operations for calculating the components of the second derivative related to the Sum 1 and Sum 2 above.

3.3 Dealing with Equal Values

The preceding observations give good motivation for trying to break the sum into three parts. However, this does not completely work as described above, because the above formula relies upon cancellation of infinite values when d_i, d_j and d_k are not all distinct (this is not just a theoretical possibility with negligible probability, because we need to consider the cases where $i = j$, $i = k$ or $j = k$). (We will use the notation $i \sim j$ to mean $d_i = d_j$, at least approximately.)

We give the formulae explicitly for the cases $i \sim j$ and $i \sim j \sim k$. The other formulae are obtained by symmetry. When $i \sim j$, the terms $\frac{e^{d_i t}}{(d_i - d_j)(d_i - d_k)} + \frac{e^{d_j t}}{(d_j - d_i)(d_j - d_k)}$ are replaced by $\frac{te^{d_i t}}{(d_i - d_k)} - \frac{e^{d_i t}}{(d_i - d_k)^2}$. When $i \sim j \sim k$, the whole sum is replaced by $\frac{t^2 e^{d_i t}}{2}$.

We have to deal with these cases separately. We have the three sums at the start of the preceding section, and each sum has to be computed for all possible cases of equality between the d_i, d_j and d_k (a total of 5 possibilities). In equality cases, two or three of the sums combine into a single sum.

Also, some of the cases are treated in exactly the same way, meaning that they can be combined into a single matrix, to increase the computational efficiency. We will define 4 matrices as follows. Table 3 summarises which of these matrices accounts for each term. [Note that the Sum 3 cases when d_k is not equal to d_i or d_j are computed in a different way, and do not need a matrix to be prepared in advance.]

Table 3: Various equality cases for second derivatives of the exponential of a matrix

	Sum 1 $\sum_k \frac{N_{ijk}e^{d_i t}}{(d_i-d_j)(d_i-d_k)}$	Sum 2 $\sum_k \frac{N_{ijk}e^{d_j t}}{(d_j-d_i)(d_j-d_k)}$	Sum 3 $\sum_k \frac{N_{ijk}e^{d_k t}}{(d_k-d_j)(d_k-d_i)}$
d_i, d_j, d_k all distinct	P	R	
$d_i = d_j$	S+P		
$d_j = d_k$	P	T+R	
$d_i = d_k$	S+P	P	S+P
$d_i = d_j = d_k$	T		

First denote $N_{ikj} = (N_\beta)_{ik}(N_\gamma)_{kj} + (N_\gamma)_{ik}(N_\beta)_{kj}$, then define

$$P_{ij} = \begin{cases} \displaystyle\sum_{k\neq i} -\frac{N_{ikj}}{(d_i-d_k)^2} & \text{if } i \sim j \\ \displaystyle\sum_{\substack{k\neq i \\ k\neq j}} \frac{N_{ikj}}{(d_i-d_j)(d_i-d_k)} - \sum_{k\sim i} \frac{N_{ikj}}{(d_i-d_j)^2} + \sum_{k\sim j} \frac{N_{ikj}}{(d_i-d_j)^2} & \text{if } i \nsim j \end{cases} \quad (1)$$

$$R_{ij} = \begin{cases} 0 & \text{if } i \sim j \\ \displaystyle\sum_{\substack{k\neq i \\ k\neq j}} \frac{N_{ikj}}{(d_i-d_k)(d_j-d_k)} + \sum_{k\sim i} \frac{N_{ikj}}{(d_i-d_j)^2} - \sum_{k\sim j} \frac{N_{ikj}}{(d_i-d_j)^2} & \text{if } i \nsim j \end{cases} \quad (2)$$

$$S_{ij} = \begin{cases} \displaystyle\sum_{k\neq i} \frac{N_{ikj}}{(d_i-d_k)} & \text{if } i \sim j \\ \displaystyle\sum_{k\sim i} \frac{N_{ikj}}{(d_i-d_k)} & \text{if } i \nsim j \end{cases} \quad (3)$$

$$T_{ij} = \begin{cases} \sum_{k\sim i} \dfrac{N_{ikj}}{2} & \text{if } i \sim j \\ \sum_{k\sim j} \dfrac{N_{ikj}}{(d_k - d_i)} & \text{if } i \nsim j \end{cases} \tag{4}$$

Now if X is the second derivative of e^{Dt}, then $w^T X s$ can be expressed as

$$\sum_{i,j} w_i s_j \frac{(N_{\beta\gamma})_{ij}(e^{d_i t} - e^{d_j t})}{d_i - d_j} + \sum_{i,j} w_i s_j \left(P_{ij} e^{d_i t} + R_{ij} e^{d_j t} + S_{ij} t e^{d_i t} + T_{ij} t e^{d_j t}\right) + \sum_{i\sim j} w_i s_j T_{ij}(t^2 - t) e^{d_j t} + \sum_k \left[\left(\sum_{i\nsim k} \frac{w_i (N_\beta)_{ik}}{d_i - d_k}\right)\left(\sum_{j\nsim k} \frac{s_j (N_\gamma)_{jk}}{d_j - d_k}\right) + \left(\sum_{i\nsim k} \frac{w_i (N_\beta)_{ik}}{d_i - d_k}\right)\left(\sum_{j\nsim k} \frac{s_j (N_\gamma)_{jk}}{d_j - d_k}\right)\right] e^{d_k t}$$

3.4 Algorithm

We now describe in full the algorithm for calculating all first and second derivatives for each site for a given tree with root R. (Choice of root does not affect the answer). Denote the vector of equilibrium frequencies by π and the diagonal matrix whose entries are π by Π. We refer to a general node as N, and its children as $N_1, \ldots, N_k$. (For a binary tree, k is 2 at internal nodes and 0 at leaf nodes.) When N is the bottom node of e, we write $e = N$. We denote the length of the branch above N by t_e.

1. Diagonalise the Q-matrix, $Q = ADA^{-1}$.
2. For each parameter β affecting the Q-matrix, calculate the derivative M_β of the Q-matrix with respect to β, and its conjugate $N_\beta = A^{-1} M_\beta A$.
3. For each pair of parameters β and γ, calculate the matrices $P_{ij}, R_{ij}, S_{ij}, T_{ij}$ defined in (1)–(4).
4. Calculate the Down and Up lists for likelihood:
   (a) For a leaf node N, $d_{N,i} = 1$ if the data at node N is i, and 0 otherwise.
   (b) The other Down lists are calculated recursively, using the formulae:

$$s_N = A^{-1} d_N$$
$$\delta_N = A e^{Dt} s_N$$
$$d_N = \delta_{N_1} \# \ldots \# \delta_{N_k}$$

[s_N and δ_N do not need to be calculated at the root node.]

(c) Calculate the Up lists for the nodes $R_1, \ldots, R_k$ directly below the root node, using the formula:

$$u_{R_i} = \delta_{R_1} \# \cdots \# \widehat{\delta_{R_i}} \# \cdots \# \delta_{R_k}$$

where $\widehat{\delta_{R_i}}$ indicates that the term δ_{R_i} is omited.

(d) Calculate all other Up lists for nodes below the root node, using the recursive formulae:

$$\begin{aligned} w_N &= A^T \Pi u_N \\ v_N &= \Pi^{-1} (A^{-1})^T e^{Dt} w_N \\ u_{N_i} &= v_N \# \delta_{N_1} \# \cdots \# \widehat{\delta_{N_i}} \# \cdots \# \delta_{N_k} \end{aligned}$$

[Note that, assuming reversibility, $\Pi^{-1}(A^{-1})^T = A$, see Appendix 1 for details.]

5. For each branch e, form the down lists $d_N^{t_e}$, $\delta_N^{t_e}$ and $s_N^{t_e}$ for every node N above e ($N > e$), and the lists $\delta_N^{t_e}$ and $v_N^{t_e}$ for the bottom node of e ($N = e$), using the recursive formulae:

$$\begin{aligned} d_N^{t_e} &= \delta_{N_1} \# \cdots \# \delta_{N_i}^{t_e} \# \cdots \# \delta_{N_k} \quad \text{where } e \leqslant N_i \\ s_N^{t_e} &= A^{-1} d_N^{t_e} \\ \delta_N^{t_e} &= \begin{cases} ADe^{Dt} s_N & \text{if } N = e \\ Ae^{Dt} s_N^{t_e} & \text{if } N > e \end{cases} \end{aligned}$$

and the formula

$$v_N^{t_e} = \Pi^{-1} (A^{-1})^T D e^{Dt} w_N \text{ (when } N = e)$$

[Again, in the reversible case, we can replace $\Pi^{-1}(A^{-1})^T$ by A. Alternatively, since all uses of $v_N^{t_e}$ involve multiplying by Π, we can directly compute $\Pi v_N^{t_e}$. We use this form here because it preserves the meaning of $v_N^{t_e}$ as $\frac{\partial v_N}{\partial t_e}$.]

6. For each parameter β affecting the Q-matrix, form the cumulative lists c_N^β, γ_N^β and the list t_N^β (defined below) using the recursive formulae:

$$\begin{aligned} c_N^\beta &= \begin{cases} 0 & \text{if } N \text{ is a leaf} \\ \sum_{i=1}^k \delta_{N_1} \# \cdots \# \gamma_{N_i}^\beta \# \cdots \# \delta_{N_k} & \text{otherwise} \end{cases} \\ t_N^\beta &= G^\beta s_N \end{aligned}$$

$$\gamma_N^\beta = A(e^{Dt}A^{-1}c_N^\beta + e^{Dt}t_N^\beta - G^\beta e^{Dt}s_N + tZs_N)$$

where

$$Z_{ij} = \begin{cases} e^{d_i t} & \text{if } i \sim j \\ 0 & \text{if } i \nsim j \end{cases} \qquad \text{and} \qquad G_{ij}^\beta = \begin{cases} \frac{(N_\beta)_{ij}}{d_j - d_i} & \text{if } i \nsim j \\ 0 & \text{if } i \sim j \end{cases}$$

[Note: the condition $i \sim j$ includes all the diagonal entries of the matrix and those off-diagonal entries for which $d_i = d_j$.]

Also calculate

$$\psi_N^\beta = G^\beta w_N$$
$$v_N^\beta = \Pi^{-1}(A^{-1})^T(e^{Dt}\psi_N^\beta - (G^\beta)^T e^{Dt}w_N + te^{Dt}w_N)$$

7. Form the first derivatives.
   (a) For each branch e, $\frac{\partial L}{\partial t_e} = \pi^T(d_M^{t_e} \# v_M)$, where M is the top node of e (or indeed any node above e). Alternatively $\frac{\partial L}{\partial t_e} = (d_N)^T(\frac{\partial e^{Qt_e}}{\partial t_e})^T \Pi u_N = s_N^T(De^{Dt_e})w_N$, where N is the bottom node of e.
   (b) For each parameter β affecting the Q-matrix, the first derivative is $\pi^T c_R^\beta$.
8. Form the second derivatives with respect to branch lengths.
   (a) For each branch e, $\frac{\partial^2 L}{\partial t_e^2} = (d_N)^T(\frac{\partial^2 e^{Qt_e}}{\partial t_e^2})^T \Pi u_N = s_N^T(D^2 e^{Dt_e})w_N$, where $e = N$.
   (b) For each pair of distinct branches with $e' < e$, let N be the bottom node of e, then $\frac{\partial^2 L}{\partial t_e \partial t_{e'}} = \left(\frac{\partial d_N}{\partial t_{e'}}\right)^T (\frac{\partial e^{Qt_e}}{\partial t_e})^T \Pi u_N = (d_N^{t_{e'}})^T(Qe^{Qt_e})^T \Pi u_N = (s_N^{t_{e'}})^T(De^{Dt_e})w_N$.
   (c) For each pair of distinct branches with $e'//e$, if N is the lowest node above both of them and $e \leqslant N_1$ and $e' \leqslant N_2$ where N_1 and N_2 are two distinct children of N, then $\frac{\partial^2 L}{\partial t_e \partial t_{e'}} = \pi^T(\delta_{N_1}^{t_e} \# \delta_{N_2}^{t_{e'}} \# ... \# \delta_{N_k} \# v_N)$. [The v_N can be omited if N is the root node.]
9. Form the second derivatives with respect to parameters affecting the Q-matrix. For each pair of parameters β and γ,

$$\frac{\partial^2 L}{\partial \beta \partial \gamma} = \sum_{e<e'} \frac{\partial^2 L}{\partial \beta_e \partial \gamma_{e'}} + \sum_{e'<e} \frac{\partial^2 L}{\partial \beta_e \partial \gamma_{e'}} + \sum_{e//e'} \frac{\partial^2 L}{\partial \beta_e \partial \gamma_{e'}} + \sum_{e=e'} \frac{\partial^2 L}{\partial \beta_e \partial \gamma_{e'}} \qquad (5)$$

We calculate this as the sum of all the following terms:

(a) the sum of the first two terms in the Equation (5) can be calculated as

$$\sum_N ((c_N^\beta)^T \Pi v_N^\gamma + (c_N^\gamma)^T \Pi v_N^\beta)$$

(b) The third term in Equation (5), i.e. the cases where $e//e'$, is obtained by summing the following terms over all nodes N which are not leaf nodes.

$$\sum_{1 \leqslant i \neq j \leqslant k} \pi^T (\delta_{N_1} \# \cdots \# \gamma_{N_i}^\beta \# \cdots \# \gamma_{N_j}^\gamma \# \cdots \delta_{N_k} \# v_N)$$

[The v_N can be omitted if N is the root node.]

(c) The sum over $e = e'$ is calculated as the following sum:

$$\sum_{i,j,N} s_{N,i} w_{N,j} \frac{(N_{\beta\gamma})_{ij}(e^{d_i t_e} - e^{d_j t_e})}{d_i - d_j} +$$
$$\sum_{i,j} \left[\left(\sum_N s_{N,i} w_{N,j} e^{d_i t} \right) P(\beta,\gamma)_{ij} + \left(\sum_N s_{N,i} w_{N,j} e^{d_j t} \right) R(\beta,\gamma)_{ij} + \right.$$
$$\left. \left(\sum_N s_{N,i} w_{N,j} t e^{d_i t} \right) S(\beta,\gamma)_{ij} + \left(\sum_N s_{N,i} w_{N,j} t e^{d_j t} \right) T(\beta,\gamma)_{ij} \right] +$$
$$\left(\sum_{i \sim j} \left(\sum_N s_{N,i} w_{N,j} (t^2 - t) e^{d_j t} \right) T(\beta,\gamma)_{ij} \right) + (t_N^\beta)^T e^{Dt} \psi_N^\gamma + (t_N^\gamma)^T e^{Dt} \psi_N^\beta$$

[If β and γ are the same parameter, some of the above sums in (a), (b) and (c) are equal, so it is possible to save a small amount of computation by only computing them once.]

10. Form the second derivatives with respect to a branch length t_e and a parameter β affecting the Q-matrix.

$$\frac{\partial^2 L}{\partial t_e \partial \beta} = \sum_{e'<e} \frac{\partial^2 L}{\partial t_e \partial \beta_{e'}} + \sum_{e'>e} \frac{\partial^2 L}{\partial t_e \partial \beta_{e'}} + \sum_{e'//e} \frac{\partial^2 L}{\partial t_e \partial \beta_{e'}} + \sum_{e=e'} \frac{\partial^2 L}{\partial t_e \partial \beta_{e'}}$$

We calculate this as the sum of the following terms:

(a) Suppose $e = N$, the first term in the equation is:

$$\sum_{e'<N} \frac{\partial^2 L}{\partial t_e \partial \beta_{e'}} = (c_N^\beta)^T \left(\frac{\partial e^{Qt_e}}{dt_e}\right)^T \Pi u_N = (c_N^\beta)^T v_N^{t_e}$$

(b) Suppose $e' = N$, the second term in the equation is:

$$\sum_{e'>e} \frac{\partial^2 L}{\partial t_e \partial \beta_{e'}} = \sum_{N>e} (d_N^{t_e})^T (\frac{\partial e^{Qt_{e'}}}{\partial \beta_{e'}})^T \Pi u_N = \sum_{N>e} (d_N^{t_e})^T v_N^{\beta}$$

(c) For each node N which is above e, suppose that $e \leqslant N_1$ where N_1 is an immediate descendent of N. The sum of following terms over all such nodes N, gives the sum of the cases where $e'//e$.

$$\sum_{2 \leq i \leq k} \pi^T (\delta_{N_1}^{t_e} \# \delta_{N_2} \# \cdots \# \gamma_{N_i}^{\beta} \# \cdots \# \delta_{N_k} \# v_N)$$

(d) The fourth term in the equation is:

$$\frac{\partial^2 L}{\partial t_e \partial \beta_e} = (d_N)^T (\frac{\partial^2 e^{Qt_e}}{\partial t_e \partial \beta_e})^T \Pi u_N = s_N^T (\frac{\partial^2 e^{Dt_e}}{\partial t_e \partial \beta_e})^T w_N =$$
$$s_N^T (\frac{\partial (De^{Dt_e})}{\partial \beta_e})^T w_N = (s_N)^T \left(N_\beta e^{Dt_e} + DN_\beta \frac{e^{d_i t_e} - e^{d_j t_e}}{d_i - d_j} \right) w_N$$

3.5 Complexity

Having explained how to find the Hessian, we now examine the complexity of the algorithm. We will use the following variables for definition of the complexity:

n	Number of rows (or columns) of the Q-matrix (61 for codon models using standard genetic code, 20 for amino acid models, 4 for nucleotide models)
b	Number of branches in the tree (or total number of nodes)
h	Height of the tree
p	Number of non-branch-length parameters
S	Number of sites

The calculated complexities for the most expensive parts of the computation are summarised in Table 4. Full details of the derivation of the complexity are in the Appendix 2. Typically, for codon or amino acid data, we have $p < n < S$ and $p < b < S$, so that the most expensive steps in Table 4 have complexity $O(b^2 nS)$ and $O(bpn^2 S)$.

For comparison, calculating only the first derivatives in the manner described above requires $O(bpn^2 S)$ computations. (It can be improved to $O((b+p)n^2 S)$.) This means that asymptotically, this method is as efficient as can reasonably be expected. Furthermore, it indicates that the method can be practically used in place of numerical methods for finding the derivatives.

Table 4: Complexity of the most expensive steps in the algorithm

Algorithm step and the calculation	Complexity
3. Calculate matrices for each pair of parameters	p^2n^3
5. Create branch-length lists	bhn^2S
6. Create parameter cumulative lists	bpn^2S
8c. Two branch lengths $e//e'$	b^2nS
9iii. Two parameters on the same branch	$(bpn^2 + bp^2n + p^2n^2)S$
10a,b,c. branch length and parameter not on the same branch	$bhpnS$
10d. branch length and parameter on the same branch	bpn^2S

4 Applications

In this section, we discuss three applications of the Hessian matrix calculation, together with comparisons to alternative methods.

As indicated earlier, both the average outer product of scores and the negative Hessian are often used as estimators of Fisher information matrix, because under some mild regularity conditions, we have

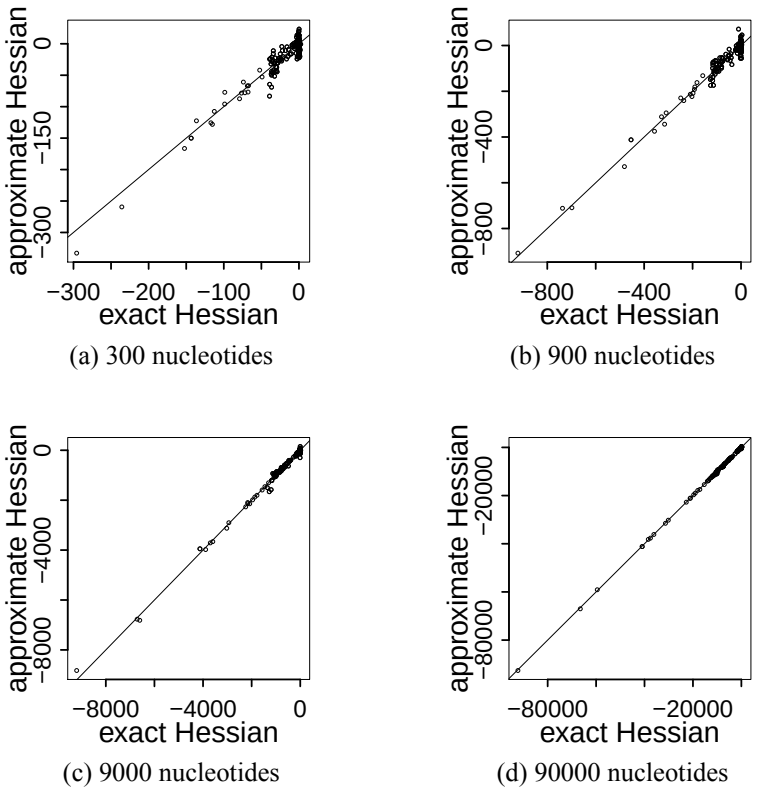
$$E\left(\frac{\partial^2}{\partial\beta\partial\beta^T}\log f(X|\beta)\right) = -E\left(\frac{\partial}{\partial\beta}\log f(X|\beta)\frac{\partial}{\partial\beta^T}\log f(X|\beta)\right), \tag{6}$$

Thus an approximation to the Hessian can be conveniently calculated:

$$\sum_i^N \frac{\partial^2}{\partial\beta\partial\beta^T}\log f(x_i|\beta) \simeq \sum_i^N \frac{\partial}{\partial\beta}\log f(x_i|\beta)\frac{\partial}{\partial\beta^T}\log f(x_i|\beta), \tag{7}$$

where the sum is taken over the sites in the data. Seo et al. (2004) have used this approximation in the fast estimation of species divergence time in a Bayesian setting where the log likelihood function was approximated by its Taylor expansion around the MLE of the parameters. They use a numerical approximation to the first derivative and use the above equation to estimate the Hessian by the outer product of the approximate first derivatives.

To study the accuracy of this approximation, we simulated several data sets with different sequence lengths based on the M_0 model. Figure 3 shows the differences between the outer product of scores approximation, based on the analytic solution of the first derivatives, and exact Hessians for the simulated data sets of various sizes. From Figure 3, we see that for very long sequences the approximation is indeed



(a) 300 nucleotides (b) 900 nucleotides

(c) 9000 nucleotides (d) 90000 nucleotides

Figure 3: Comparison of exact and approximate Hessians for various sequence lengths.

very good. However, for shorter sequences, the comparison is much less good. It is important to note that while the errors are small compared with the overall size of the matrix, a lot of entries have large relative errors, and the same is true for the inverse of the Hessian matrix. We will compare the exact Hessian matrix and the approximate Hessian from (7) based on the analytic solution of the first derivatives, for the following applications, and compare both of them with existing methods.

4.1 Optimisation Using the Newton-Raphson Method

The Newton-Raphson method is a popular method for non-linear optimisation. It is based on a quadratic approximation obtained using the Hessian matrix and the gradient vector. In the absence of Hessian calculation, optimisation algorithms must either use numerical approximations which are often hampered by numerical preci-

sion issues, or use an alternative method, such as Quasi-Newton methods. A number of software packages for phylogenetic analysis have used Quasi-Newton methods. Through the following time trials, we show that the Newton-Raphson method based on our Hessian calculation provides an efficient alternative for the optimisation of phylogenetic likelihood functions.

We ran our software COLD and a popular software PAML (Yang, 2007), which uses a quasi-Newton method, to compare convergence times on a range of data sets, distributed with the PAML package, using the model M_0 (Goldman, Yang 1998). It is important to note that this is a comparison between two different implementations of the different methods. There can be a large difference between two implementations of the same method, so the results of this time trial should not be taken as a thorough comparison between quasi-Newton methods and Newton-Raphson methods. There are various factors which give one piece of software an advantage over another in this time trial. PAML has been developed many years ago, and has undergone many years of fine-tuning, so it would be expected to include more optimisations not yet implemented in COLD. PAML is a very ad-hoc software package — it is designed to fit models based on M_0, so it can assume that the model in question is M_0, and make appropriate optimisations (for example, no simultaneous multiple-nucleotide changes), while COLD is designed to be more general software, for fitting a wide class of models, and therefore is more limited in the optimisations available. The data sets were distributed with PAML, so they are data sets on which PAML has been developed, thus optimisations that are important for these data sets are more likely to be included in PAML. On the other hand, the time trials were run on the machine on which COLD was developed, and the programs were compiled using the same compiler that was used in developing COLD, so some optimisations in COLD may be specific (or at least most important) to that machine setup.

We also include a comparison with a Newton-Raphson method using the approximate Hessian given by (7). We do not know of any software that uses this method for Newton-Raphson optimisation, but the potential is clear. The approximate Hessian method was coded specifically for this comparison, and optimisation was using the same Newton-Raphson routine as COLD, which may not be the most appropriate. To make fair comparisons, we ran COLD in single-threaded mode, although it permits multi-threaded (parallel) execution, and thus it can run faster than indicated here on modern computers.

Table 5 gives the median running times from 3 runs of the same starting values for the three programs on a range of data sets, with each data set evaluated with each program for two different starting values, near to the optimum. (Details of the time trial are in the supplemental material). All programs converged to the same log-likelihood values at the same MLE. We see that the times are comparable, and

there are several cases where the Newton-Raphson method greatly speeds up optimisation. This clearly demonstrates the potential of the Newton-Raphson method.

Comparing exact Hessian and approximate Hessian calculations, the Newton-Raphson method based on exact Hessian generally converges in very few steps, but in some cases this benefit in terms of number of steps until convergence is not sufficient to justify the computation time needed to calculate an exact Hessian at each step. The Newton-Raphson method based on approximate Hessian needs many more steps to converge, but is very fast calculating the approximate Hessian at each step, since it only involves calculating first derivatives for each site. This advantage especially shows in the cases of longer sequences where the Hessian approximation is close enough to the Hessian. However, the accuracy is limited for shorter genes, so the number of steps required (and therefore total time) can increase significantly. In order to gain the benefits of both methods in the appropriate cases, it should be possible to develop a hybrid method which occasionally performs an exact Hessian calculation, and uses approximations the rest of the time. Possibly, computing approximate Hessian matrices in the early stages of the optimisation where the quadratic approximation is less accurate and thus errors in the approximation are less serious, and computing the exact Hessian in later stages when it can lead to quicker convergence. We hope that in near future more efficient implementations of the Newton-Raphson method will be available.

4.2 Inference

The availability of Hessian matrix allows us to use likelihood theory for inference for the model parameters. Previously, inference for parameter estimates obtained by maximising the likelihood function in phylogenetic analysis has mostly been based on bootstrap methods, which are very time consuming. The Hessian matrix evaluated at the MLE provides an estimate for the asymptotic variance of the parameter estimates, which we now use to calculate confidence intervals (CIs) for the parameter estimates.

We demonstrate the CIs calculated by the exact Hessian (Hessian method) on simulated data sets, comparing with CIs obtained using non-parametric bootstraps (Bootstrap method) and using the outer product of scores approximation to the asymptotic variance (Scores method) given by (7).

We simulated two scenarios with different sequence lengths based on a 10-taxon tree. The tree topology used for the simulation is from Bielawski and Yang (2005) with the out group removed, as shown in Figure 4. The first scenario has sequence length 100 codons, and the second has sequence length 300 codons. We simulated 100 data sets in each scenario. The M_0 model is used to simulate and analyse the data on the same tree topology, so there is no model misspecification. We calcu-

Table 5: Median running times (from 3 runs) for running COLD, PAML and COLD with the approximate Hessian calculation using the Scores method, on various data sets. (Full details in supplementary material.)

Data Set	Taxa	Sites	COLD	PAML	Approximate Hessian
mtCDNAmam	20	3331	7:41	10:26	7:02
			8:30	10:09	7:10
mtCDNApri	7	3331	2:21	0:46	0:55
			2:07	0:54	0:48
MouseLemurs	35	604	3:14	12:02	2:42
			2:56	10:28	2:39
LysozymeSmall	7	130	0:07	0:04	0:05
			0:06	0:06	0:05
HIVenvSweden	13	91	0:09	0:09	0:25
			0:10	0:10	0:17
Lysin	26	122	0:33	2:00	1:48
			0:33	2:01	1:20

lated 95% CIs for each of 17 branch lengths using each method. The CIs based on likelihood theory are symmetric about the MLEs. The non-parametric bootstrap CIs are calculated based on 100 bootstrap replicates from each simulated data set and these CIs are not necessarily symmetric about MLEs. The empirical coverage rate was calculated as the percentage of times the true branch length was covered by the CIs for each branch length. The standard error for each coverage rate is $\sqrt{(.95)(.05)/100} = 0.0218$, so a statistically significant difference from the nominal coverage rate of 0.95 would be 0.0436. The mean and standard deviation of the width of the CIs for each branch length are also calculated over 100 simulations. We show these results in Figure 5.

From Figure 5, we can see that the CIs calculated by the Hessian and bootstrap methods are very close according to both the coverage and mean width of CIs. The standard deviation of the width of CIs obtained using the bootstrap method is larger than that obtained using the Hessian method. This can be improved by increasing the bootstrap replicates. However the Hessian method uses only a fraction of the time of the bootstrap method to calculate these CIs even with the replicate number set as only 100. It is also noted that the joint confidence regions (CRs) of multiple parameters or simultaneous testing for multiple parameters are also directly avail-

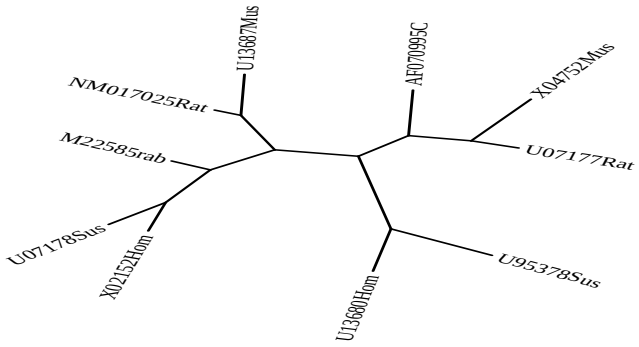


Figure 4: Tree used for simulations.

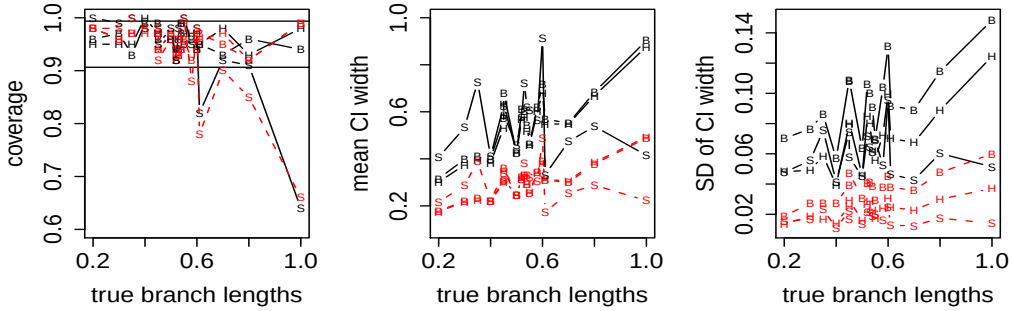


Figure 5: The comparisons of CIs constructed by Hessian (H), Scores (S) and bootstrap (B) methods over the coverage, average width of CIs and standard deviation (SD) of the width of CIs.

able from the Hessian method, but it would require significantly more bootstrap replicates to obtain comparable accuracy for such CRs or simultaneous testing.

The coverage rates for the CIs obtained by the Scores method for shorter branch lengths are within the range of $0.95 \pm 2\text{SE}$ (shown by the two horizontal lines in the plot). But the coverage rates for the Scores method are significantly lower than the nominal level for the longer branch lengths. The mean widths of CIs for the Scores method are larger when the true branch lengths are short, and much smaller than that of other methods when the true branch lengths are longer. These results are consistent because the standard deviation of the width of CIs for the Scores method are all relatively small. It is clear that the Scores method does not yield inference results that are comparable to the other two methods. This is consistent with the observations in Porter (2002), where various estimators for Fisher information and the resulting confidence intervals are compared in a regression setting and the observed information matrix (Hessian method) was found to perform better than the Scores method. For inference, the gain in computation speed from using the Scores method has little benefit, since the time required to perform a full Hessian calculation at the MLE is negligible compared to the time needed to optimise the dataset.

It is worth noting that the Scores method is quite popular. It has been implemented in the software `multidivtime` for studying rates of molecular evolution and for estimating divergence times (Thorne, Kishino & Painter, 1998). The MCMC implementation there requires many evaluations of likelihood for different parameter values. To make the computation feasible, the log-likelihood surface is approximated with a multivariate Gaussian distribution centered at the MLE of the parameters and with covariance matrix obtained using the Scores method. The Hessian calculation presented in this paper should be able to improve the likelihood approximation in the software `multidivtime`, especially when the sequence lengths are relatively small.

4.3 Local Influence Analysis

Whenever we make inferences from statistical models and data, it is important to know whether the inferred results are supported by the majority of the data, or are caused mainly by a few extreme observations. Influence analysis measures the extent to which each observation affects the overall results. This can be used to identify data points which do not satisfy the model assumptions. These data points may lead us to important biological conclusions. For example, if some sites within a gene are under positive selection, then when we analyse the data under a model without rate variation among sites, these sites will have a large positive influence on the estimated value of the nonsynonymous/synonymous ratio ω. We will give an example of this use of influence analysis.

There are a number of different methods for influence analysis. Traditionally, influence analysis in Phylogeny has been performed using deletion influence, which involves removing part of the data and re-analysing the remainder to determine how much influence the removed data has. This process can be prohibitively time consuming — for every site whose influence we want to estimate, we need to perform a new optimisation.

Local influence analysis examines the effect on parameter estimates of small changes to the data. It does this by viewing the parameter estimates as a function of the data, and taking the derivative of this function. For continuous data, there are obvious ways in which we can view the parameter estimates as a differentiable function of the data. For the discrete data in phylogeny, we need to extend the possible data points into a continuous space. One easy way to do this is to assign a weight to each site of the data, which represents the frequency with which that site pattern was observed (so a weight of 1 represents a single occurance of that site pattern, a weight of 2 represents two occurances, and so on). Deletion influence considers the effect of changing the weight of a particular site pattern to zero. The overall log-likelihood is given by the log-likelihood of each site pattern multiplied by the corresponding weight. We can obtain a continuous data space by using non-integer weights. That is, even though we cannot observe a particular site pattern $\frac{3}{2}$ times, we can still calculate the likelihood and parameter estimates under the assumption that we did observe the pattern this many times. Now the parameter estimates are a differentiable function of the weights of each site, and the derivative of a parameter estimate with respect to the weight of a particular site is a measure of the influence of that site on that parameter estimate.

Let w_i represent the weight of site i. Let θ be the vector of parameters that we will optimise. The log-likelihood l is a function of both the weights w_i and the parameters θ. The MLE is a function $\hat{\theta}(w_i)$ of the weights, satisfying $\frac{\partial l}{\partial \theta}\big|_{\hat{\theta}(w_i)} = 0$. Differentiating both sides of this equation with respect to w_i, we get that

$$\frac{\partial^2 l}{\partial \theta \partial \theta^T}\frac{\partial \hat{\theta}}{\partial w_i} + \frac{\partial^2 l}{\partial \theta \partial w_i} = 0, \tag{8}$$

holds when the derivatives are evaluated at $\hat{\theta}(w_i)$. We rearrange this to get:

$$\frac{\partial \hat{\theta}}{\partial w_i} = -\left[\frac{\partial^2 l}{\partial \theta \partial \theta^T}\right]^{-1}\left[\frac{\partial^2 l}{\partial \theta \partial w_i}\right] \tag{9}$$

The first term in the above result is the Hessian matrix and the second term is the first derivative of the site log-likelihood with respect to the parameters (since $\frac{\partial l}{\partial w_i}$ is just the log-likelihood of site i). Therefore, our Hessian calculation allows us to calculate the local influence of a site.

As an example of the application of this, we use influence analysis to detect sites under positive selection. We simulated 100 data sets consisting of 300 codon sites each. Of these 300 sites, the first 30 were under positive selection ($\omega = 2$). The next 10 were under approximately neutral selection ($\omega = 0.8$), and the other 260 were under negative selection ($\omega = 0.03$). We then calculated the maximum likelihood estimator under model M_0, and the local influence of each site on the estimate of ω. The local influence function for a typical data set is shown in the left panel of Figure 6. From this figure, we can clearly see that our influence analysis has detected most of the sites under positive selection.

The standard method for detecting positive selection is to fit a mixture model such as M_2 (Yang et al, 2000) and use a likelihood ratio test to compare with a model without positive selection. (Note that M_2 is slightly misspecified in our simulated data sets, because it assumes a class of sites under exactly neutral selection, rather than approximately neutral selection as was simulated.) One way in which our influence analysis method has a clear advantage over a mixed model is in computation time. Once the parameters have been optimised for a simple model, performing influence analysis requires only a single Hessian calculation, whereas the mixture model method requires a completely new optimisation, with a more complicated model. This means that our influence analysis is valuable as a diagnostic tool for quickly deciding whether a particular data set merits more detailed investigation.

The right panel of Figure 6 shows the local influence of the sites against the posterior probability of positive selection for all sites from a random sample of 20 data sets out of 100 simulated data sets. (The data from all 100 data sets shows a similar pattern, but is more difficult to see because the figure is more cluttered. The analyses from all 100 simulated data sets are available in the supplementary materials.) From this figure, it is not clear whether the influence analysis or the mixed model provides better results in terms of classification of sites. However, since the two methods provide different classifications, the influence analysis is providing additional information not available from the mixture model analysis. Therefore, we expect that combining the information from the two methods will give us a more accurate classification of sites under positive selection. The question of how best to combine the information from both analyses is an ongoing project of the authors, and will hopefully be published soon, along with other applications of local influence analysis in phylogeny, such as detecting heterotachy in ω (or other parameters).

5 Conclusions and Future Work

We have presented a method for calculating the second derivatives of likelihood for phylogeny, with respect to a wide range of parameters affecting the rates of substitu-

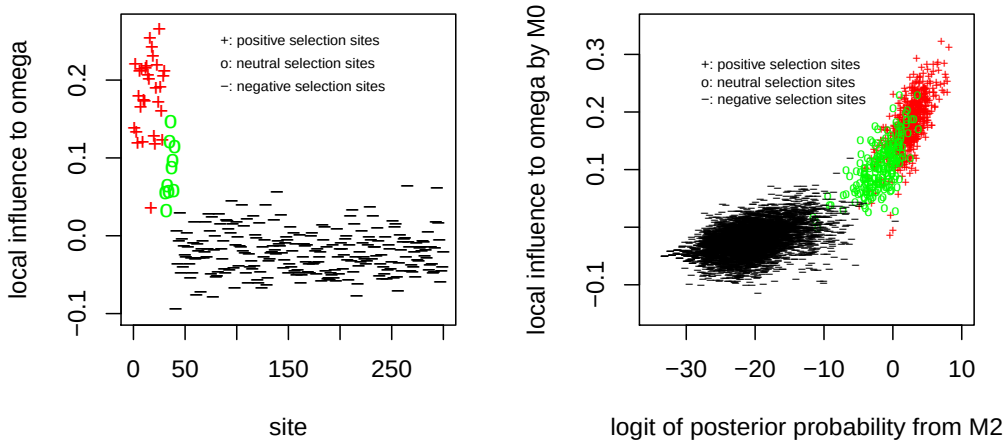


Figure 6: Influence of each site on the estimate of ω for one data set (left) and for 20 data sets randomly sampled from 100, against the logit transformation of posterior probability of positive selection under model M_2 (right).

tions and branch lengths. We then developed methods improving the computational efficiency of this calculation. The complexity analysis of the algorithm presented in Section 3.5 demonstrates that the program scales well to larger trees. The method presented here can be applied to DNA, amino acid or codon models. Indeed the method could also be applied to other Markov chain models. This computation is implemented in our software COLD, which is available from the first author's website at `http://www.mathstat.dal.ca/~tkenney/Cold/`
In future work, we plan to extend this implementation to wider classes of models (see Section 2.1 for details of the assumptions that we hope to easily relax).

This Hessian calculation has the potential to be applied in a number of ways in Phylogeny. Firstly, it allows us to use the Newton-Raphson method to maximise likelihood. It is well known that with a good starting value, the Newton-Raphson method converges very fast. Parsimonious methods can usually provide reasonably good starting values. Thus the Newton-Raphson method can be implemented in a wide variety of models in phylogeny. Fast computation can also help with tree-search problems. In future work, we plan to study this optimisation issue in more detail, and combine our Hessian calculation with the faster approximations such as the Scores method, in order to get the benefits from both methods.

The availability of the Hessian matrix calculation allows us to use likelihood theory for inference for the model parameters. Previously, inference for parameter estimates obtained by maximising the likelihood function in phylogenetic analysis has mostly been based on bootstrap methods, which are very time consuming. We demonstrated that, for confidence intervals, using the approximate Hessian calculation by the Scores method does not provide the same accuracy as using the exact Hessian. In future work, we plan to fully develop the potential of the Hessian matrix in different kinds of inference problems in phylogeny, and we also plan to obtain a calculation of the Fisher information matrix, to be used for inference in phylogeny.

The availability of the Hessian matrix also allows us to apply local influence methodology to explore whether the model-data agreement support the model assumptions, or whether the analysis results are sensitive to the model and/or data. These diagnostic results can be useful for detecting sites, or potentially lineages, which do not fit the model assumptions. The insight gained in such analysis can lead us to significant model improvement and/or biological conclusions, as seen in the positive selection application in this paper.

Appendix 1: Differentiating the P Matrix

Theorem 1. *Suppose parameters β and γ are two parameters which influence the Q matrix, and the following derivatives are readily available: $\frac{\partial Q}{\partial \beta} = M_\beta$, $\frac{\partial Q}{\partial \gamma} = M_\gamma$ and $\frac{\partial^2 Q}{\partial \beta \partial \gamma} = M_{\beta\gamma}$. Then we have:*

1. *If the process is reversible, then for any parameter values, Q is diagonalisable as $Q = ADA^{-1}$.*

2. *For any invertible constant matrix C, we define $X = C^{-1}QC$, so that $Q = CXC^{-1}$, and we have that*

$$\frac{\partial P}{\partial \beta} = \frac{\partial}{\partial \beta}\left(e^{Qt}\right) = C\frac{\partial (e^{Xt})}{\partial \beta}C^{-1},$$

and we define $N_\beta = \frac{\partial X}{\partial \beta} = C^{-1}M_\beta C$. Now, for a given set of parameter values θ_0, if $Q(\theta_0)$ is diagonalizable as $Q(\theta_0) = ADA^{-1}$, then by choosing $C = A$ in the above expression, $X(\theta_0)$ is equal to the diagonal matrix D with entries d_i, and the ij^{th} entry of the matrix $\left.\frac{\partial (e^{Xt})}{\partial \beta}\right|_{\theta_0}$ is given by

$$\left(\left.\frac{\partial (e^{Xt})}{\partial \beta}\right|_{\theta_0}\right)_{ij} = \begin{cases} \frac{(N_\beta)_{ij}(e^{d_i t} - e^{d_j t})}{d_i - d_j} & \text{if } d_i \neq d_j \\ (N_\beta)_{ij} t e^{d_i t} & \text{if } d_i = d_j \end{cases}$$

Note that the above result for $d_i = d_j$ is also the limit of the fractional form for $d_i \neq d_j$, when $d_j \to d_i$, as can be seen e.g. by using L'Hopîtal's rule.

3. *Furthermore, denoting $N_\gamma = A^{-1}M_\gamma A$ and $N_{\beta\gamma} = A^{-1}M_{\beta\gamma}A$, the second derivative of the transition matrix P with respect to parameters β and γ (which could be the same parameter) can be written as*

$$\left.\frac{\partial^2 P}{\partial\beta\partial\gamma}\right|_{\theta_0} = A\left.\frac{\partial^2(e^{Xt})}{\partial\beta\partial\gamma}\right|_{\theta_0} A^{-1},$$

where the ij^{th} entry of matrix $\left.\frac{\partial^2(e^{Xt})}{\partial\beta\partial\gamma}\right|_{\theta_0}$ is given by the following, with appropriate limiting values taken in the cases when $d_i = d_j$, $d_i = d_k$, or $d_j = d_k$:

$$\left(\left.\frac{\partial^2(e^{Xt})}{\partial\beta\partial\gamma}\right|_{\theta_0}\right)_{ij} = \frac{(N_{\beta\gamma})_{ij}(e^{d_i t} - e^{d_j t})}{d_i - d_j} + \sum_k \left((N_\beta)_{ik}(N_\gamma)_{kj} + (N_\gamma)_{ik}(N_\beta)_{kj}\right) h_{ijk}$$

where $h_{ijk} = \frac{e^{d_i t}}{(d_i-d_j)(d_i-d_k)} + \frac{e^{d_j t}}{(d_j-d_i)(d_j-d_k)} + \frac{e^{d_k t}}{(d_k-d_j)(d_k-d_i)}$.

Proof. 1. By the symmetry condition, we have that $Q = R\Pi$, where R is symmetric and Π is diagonal and R and Π are non-negative definite. We can rewrite this as $Q = \Pi^{-\frac{1}{2}}(\Pi^{\frac{1}{2}}R\Pi^{\frac{1}{2}})\Pi^{\frac{1}{2}}$. Since the middle composite is real symmetric, it is diagonalisable, say $(\Pi^{\frac{1}{2}}R\Pi^{\frac{1}{2}}) = HDH^{-1}$ for some orthogonal matrix H and diagonal matrix D. Thus we have that Q is diagonalisable as $Q = ADA^{-1}$, where $A = \Pi^{-\frac{1}{2}}H$ is the matrix of left eigenvectors of Q, and D is the diagonal matrix of eigenvalues.

2. The first statement is obvious. [Note: The property $e^{CXC^{-1}} = Ce^XC^{-1}$ used in that statement is true for any two matrices C and X.]

We want to find $\left.\frac{\partial(e^{Xt})}{\partial\beta}\right|_{\theta_0}$ in the case where $X(\theta_0)$ is a diagonal matrix D with entries d_i. By definition, $e^{Xt} = \sum_{n=0}^{\infty}\frac{X^n t^n}{n!}$, and by the product rule, $\left.\frac{\partial(X^n)}{\partial\beta}\right|_{\theta_0} = \sum_{k=1}^{n} D^{k-1}N_\beta D^{n-k}$. Now since D is diagonal with entries d_i, we have that

$$\left(\left.\frac{\partial(X^n)}{\partial\beta}\right|_{\theta_0}\right)_{ij} = \sum_{k=1}^{n} d_i^{k-1}(N_\beta)_{ij}d_j^{n-k} = \begin{cases} (N_\beta)_{ij}\frac{d_i^n - d_j^n}{d_i - d_j} & \text{if } d_i \neq d_j \\ n(N_\beta)_{ij}d_i^{n-1} & \text{if } d_i = d_j \end{cases}$$

Summing over n, we therefore get

$$\left(\left.\frac{\partial(e^{Xt})}{\partial\beta}\right|_{\theta_0}\right)_{ij} = \sum_{n=0}^{\infty}\frac{t^n}{n!}\left(\left.\frac{\partial(X^n)}{\partial\beta}\right|_{\theta_0}\right)_{ij} = \begin{cases} \frac{(N_\beta)_{ij}(e^{d_i t} - e^{d_j t})}{d_i - d_j} & \text{if } d_i \neq d_j \\ (N_\beta)_{ij}te^{d_i t} & \text{if } d_i = d_j \end{cases}$$

Note that the above result for $d_i = d_j$ is also the limit of the fractional form for $d_i \neq d_j$, when $d_j \to d_i$, as can be seen e.g. by using L'Hopîtal's rule.

3. Now we consider the second derivative $\left.\frac{\partial^2 P}{\partial\beta\partial\gamma}\right|_{\theta_0}$. Similar to the first derivative, we have $\left.\frac{\partial^2 P}{\partial\beta\partial\gamma}\right|_{\theta_0} = A \left.\frac{\partial^2(e^{Xt})}{\partial\beta\partial\gamma}\right|_{\theta_0} A^{-1}$. There are now 3 types of terms to consider when we apply the product rule twice to calculate $\left.\frac{\partial^2(X^n)}{\partial\beta\partial\gamma}\right|_{\theta_0}$, when again $X(\theta_0)$ is a diagonal matrix D:

   (a) terms where $\frac{\partial}{\partial\beta}$ happens to an earlier factor than $\frac{\partial}{\partial\gamma}$. For example, $\frac{\partial X}{\partial\beta}\frac{\partial X}{\partial\gamma}$.

   (b) terms where $\frac{\partial}{\partial\beta}$ happens to a later factor than $\frac{\partial}{\partial\gamma}$. For example, $\frac{\partial X}{\partial\gamma}\frac{\partial X}{\partial\beta}$.

   (c) terms where both $\frac{\partial}{\partial\beta}$ and $\frac{\partial}{\partial\gamma}$ happen to the same factor. For example $\frac{\partial^2 X}{\partial\beta\partial\gamma}$.

   The calculation in the third case is like the first derivative calculation shown earlier: we just replace N_β by $N_{\beta\gamma}$. The first two cases are symmetric, so we only study the first. In this case $\left.\frac{\partial^2(X^n)}{\partial\beta\partial\gamma}\right|_{\theta_0} = \sum_{0\leqslant a\leqslant b\leqslant n-2} D^a N_\beta D^{b-a} N_\gamma D^{n-2-b}$. Thus

$$\left(\left.\frac{\partial^2(X^n)}{\partial\beta\partial\gamma}\right|_{\theta_0}\right)_{ij} = \sum_{0\leqslant a\leqslant b\leqslant n-2}\sum_k d_i^a (N_\beta)_{ik} d_k^{b-a} (N_\gamma)_{kj} d_j^{n-2-b} = \sum_k (N_\beta)_{ik}(N_\gamma)_{kj} \sum_{0\leqslant a\leqslant b\leqslant n-2} d_i^a d_k^{b-a} d_j^{n-2-b}$$

   If we let $b' = b - a$, the inner sum becomes

$$s_{ijk}^n = \sum_{0\leqslant a\leqslant b\leqslant n-2} d_i^a d_k^{b-a} d_j^{n-2-b} = \sum_{0\leqslant a\leqslant n-2} d_i^a \sum_{0\leqslant b'\leqslant n-2-a} d_k^{b'} d_j^{n-2-a-b'}$$

   Then we have a sum over b', which evaluates to $\frac{d_j^{n-1-a} - d_k^{n-1-a}}{d_j - d_k}$ when $d_j \neq d_k$, so we get $s_{ijk}^n = \sum_{0\leqslant a\leqslant n-2} d_i^a \frac{d_j^{n-1-a} - d_k^{n-1-a}}{d_j - d_k}$. Now, if we let $a = n-1$, then $d_j^{n-1-a} - d_k^{n-1-a} = 0$, so we have

$$s_{ijk}^n = \sum_{0\leqslant a\leqslant n-1} d_i^a \frac{d_j^{n-1-a} - d_k^{n-1-a}}{d_j - d_k} = \frac{1}{d_j - d_k}\left(\sum_{a=0}^{n-1} d_i^a d_j^{n-1-a}\right) + \frac{1}{d_k - d_j}\left(\sum_{a=0}^{n-1} d_i^a d_k^{n-1-a}\right)$$

   By evaluating each of the sums, and adding the results together, we get (for d_i, d_j, and d_k distinct)

$$s_{ijk}^n = \frac{d_i^n}{(d_i - d_j)(d_i - d_k)} + \frac{d_j^n}{(d_j - d_i)(d_j - d_k)} + \frac{d_k^n}{(d_k - d_j)(d_k - d_i)}$$

with the appropriate limiting values taken when two of them are equal. Thus the sum over the first type of terms for the derivative $\left(\frac{\partial^2(e^{Dt})}{\partial\beta\partial\gamma}\Big|_{\theta_0}\right)_{ij}$ is

$$\sum_k (N_\beta)_{ik}(N_\gamma)_{kj}h_{ijk}$$

where

$$h_{ijk} = \left(\frac{e^{d_i t}}{(d_i-d_j)(d_i-d_k)} + \frac{e^{d_j t}}{(d_j-d_i)(d_j-d_k)} + \frac{e^{d_k t}}{(d_k-d_j)(d_k-d_i)}\right)$$

with appropriate limiting values taken in the cases when $d_i = d_j$, $d_i = d_k$, or $d_j = d_k$.

□

Note: when we actually want to numerically calculate this, taking rounding errors into account, we need to decide how close d_i and d_j should be for us to treat them as equal, and use the limiting form. To do this, we consider the errors caused by rounding and by using the limiting form. The relative error caused by using the limiting form is approximately $\frac{d_i-d_j}{2}$, since we can neglect terms after the second in the Taylor series. Meanwhile the absolute rounding error induced when we use the general formula is about the machine epsilon, which we will denote ϵ, that is, the relative error caused by rounding off numbers in the machine, or $\epsilon|d_i|$, whichever is larger. This means that the relative error in the general formula is about $O\left(\frac{(1+|d_i|)\epsilon}{d_i-d_j}\right)$. We should therefore use the formula for $d_i = d_j$ whenever $(d_i-d_j)^2 < \epsilon(1+|d_i|)$.

For details about how to compute these derivatives more efficiently, see Section 3.2.

Appendix 2: Detailed Complexity Analysis

We calculate the complexity of the various steps in the algorithm. Recall the meanings of all relevant variables:

n	Number of rows (or columns) of the Q-matrix
v	Number of internal nodes in the tree
b	Number of branches in the tree (or total number of nodes)
h	Height of the tree
p	Number of non-branch-length parameters
S	Number of sites

1. The typical complexity of diagonalising a symmetric matrix (and we can diagonalise the Q-matrix by first diagonalising the symmetric matrix $R = \Pi^{\frac{1}{2}} Q \Pi^{-\frac{1}{2}}$) is $O(n^3)$ (depending on the method used and the desired accuracy). This only needs to be calculated once.

2. For most of the parametrisations that have been considered, differentiating the Q-matrix with respect to a parameter is not expensive. However, to calculate N_β, we need to conjugate the derivative by A. This requires $O(n^3)$ operations. However, this is only necessary once for each parameter, for a total complexity $O(pn^3)$.

3. For each pair β, γ of Q-matrix parameters, calculating the matrices $P(\beta,\gamma)$, $R(\beta,\gamma)$, $S(\beta,\gamma)$, and $T(\beta,\gamma)$, requires $O(n^3)$ operations. Therefore calculating these matrices for all pairs of Q-matrix parameters requires $O(p^2n^3)$ operations.

4. For each branch, multiplying a vector by e^{Qt} (expressed as a product of 3 matrices) requires $O(n^2)$ operations. To create the likelihood lists, we need to do this for each branch, so we need $O(bn^2)$ operations for each site. Creating the up lists also requires $O(bn^2)$ operations for each site.

5. For the derivatives of lists with respect to branch lengths, the recursion along each branch takes $O(n^2)$ operations, and we need to do this for the derivatives with respect to all lower branches, so the total number of times we need to do this is the number of comparable pairs of branches. This is $O(bh)$ (for each branch, there are on average $O(h)$ branches above that branch) so that in total, computing the derivatives of the lists with respect to branch lengths takes $O(bhn^2)$ operations.

6. Forming the cumulative lists for derivatives with respect to a parameter β requires $O(bn^2)$ operations. Therefore in total, computing the lists for all parameters requires $O(bpn^2)$ operations.

7. To calculate each first derivative, we need to combine a pair of lists (a down list and an up list for branch lengths, and a cumulative list for other parameters), which takes $O(n)$ operations. We need to do this for each parameter, so it requires $O(bn)$ operations for the branch lengths and $O(pn)$ operations for the other parameters.

8. For second derivatives with respect to branch lengths:

(a) Calculating the second derivative with respect to a single branch length requires calculating $s_N^T(D^2e^{Dt_e})w_N$, which can be done in $O(n)$ operations, and has to be calculated for each branch and each site.

(b) For a pair of branches with one above another, $e < e'$, with N the bottom node of e', computing $(s_N^{e'})^T(De^{Dt_e})w_N$ requires $O(n)$ operations. Therefore, calculating these terms for all pairs of branches at a given site requires $O(bhn)$ operations.

(c) For a pair of branches $e//e'$, the second derivative can be calculated in $O(n)$ operations. This needs to be done for each pair of branches and each site, so for each site $O(b^2n)$ operations are required.

9. For each pair of parameters β and γ:

(a) Calculating the sum over e' of cases where β and γ act on comparable branches $e' < e$ requires $O(n)$ operations. These sums then have to be summed over all branches e whose bottom node is not a leaf. Having to do this for all pairs of parameters therefore requires $O(vp^2n + vpn^2)$ operations for each site.

(b) Calculating the sum of cases where β and γ act on branches $e//e'$ requires $O(n)$ operations at each internal node. (Actually, slightly more are needed in the case of a multifurcating tree, though for a multifurcating tree, there are fewer internal nodes). When this is done for each internal node and each pair of parameters , it gives a total of $O(vp^2n)$ operations for each site.

(c) We calculate the effect to the second derivative where β and γ act on the same branch as the following sum:

$$\sum_{i,j,N} s_{N,i}w_{N,j}\frac{(N_{\beta\gamma})_{ij}(e^{d_it_e}-e^{d_jt_e})}{d_i-d_j} + \sum_{i,j}\left[\left(\sum_N s_{N,i}w_{N,j}e^{d_it}\right)P(\beta,\gamma)_{ij} + \left(\sum_N s_{N,i}w_{N,j}e^{d_jt}\right)R(\beta,\gamma)_{ij} + \left(\sum_N s_{N,i}w_{N,j}te^{d_it}\right)S(\beta,\gamma)_{ij} + \left(\sum_N s_{N,i}w_{N,j}te^{d_jt}\right)T(\beta,\gamma)_{ij}\right] + \left(\sum_{i\sim j}\left(\sum_N s_{N,i}w_{N,j}(t^2-t)e^{d_jt}\right)T(\beta,\gamma)_{ij}\right) + (t_N^{\beta})^Te^{Dt}\psi_N^{\gamma} + (t_N^{\gamma})^Te^{Dt}\psi_N^{\beta}$$

over all edges of the tree. The terms $\sum_e s_iw_je^{d_it}$, $\sum_e s_iw_je^{d_jt}$, $\sum_e s_iw_jte^{d_it}$, and $\sum_e s_iw_jte^{d_jt}$, each take $O(b)$ operations to compute for all i and j, and

we need to compute a total of $O(n^2)$ such terms. Therefore, computing all these terms requires $O(bn^2)$ operations for each site. Combining them with the matrices $P(\beta,\gamma)$, etc. requires $O(n^2)$ operations, and has to be done for each pair of parameters, leading to a total of $O(p^2n^2)$ operations. Since the matrices $P(\beta,\gamma)$, etc. are the same for all sites, this term does not need to be repeated for each site. The terms $(t(\beta)_N)^T e^{Dt}\psi(\gamma)_N$ and $(t(\gamma)_N)^T e^{Dt}\psi(\beta)_N$ take $O(n)$ operations for each branch and each pair of parameters, and so require $O(bp^2n)$ operations for each site.

10. For the derivative with respect to one branch length and one Q-matrix parameter:

    (a) Computing $(c_N(\beta))^T v_N(e)$ for each parameter and each branch, requires a total of $O(hbpn)$ operations on each site.

    (b) Computing $\sum_{N>e}(d_N^e)^T v_N(\beta)$ for a parameter β and a branch e, requires $O(hn)$ operations. Thus a total of $O(bhpn)$ operations are needed on each site.

    (c) The case where the branch and the parameter are on branches $e//e'$ is calculated as the sum

    $$\sum_{2\le i\le k} \pi^T(\delta^e_{N_1}\#\delta_{N_2}\#\cdots\#\gamma_{N_i}(\beta)\#\cdots\#\delta_{N_k}\#v_N)$$

    which requires $O(nh)$ operations for each parameter and each branch, on each site. Thus, a total of $O(hbpn)$ operations on each site.

    (d) The case where the parameter is differentiated on the branch is calculated as $s_N^T(\frac{\partial(De^{Dt_e})}{\partial\beta_e})^T w_N$, which requires $O(n^2)$ operations for each parameter and each branch, and thus a total of $O(bpn^2)$ operations on each site.

References

Ababneh F., Jermiin L. S., Robinson J. 2006. Generation of the exact distribution and simulation of matched nucleotide sequences on a phylogenetic tree. *Journal of Mathematical Modelling and Algorithms* 5: 291-308.

Bickel P. J., Doksum K. A. 2001. *Mathematical Statistics: Basic Ideas and Selected Topics* (2nd Edition) **VOL I** P. 180-181, P. 335, P.399-400.

Bielawski J. P., Yang Z. 2005 Maximum likelihood methods for detecting adaptive protein evolution. *Statistical Methods in Molecular Evolution.* Editor: Nielsen, R. Springer-Verlag: New York. p. 103-124.

Bryant D., Galtier N., Poursat M. A. 2005. Likelihood calculation in molecular phylogenetics. In *Mathematics of Evolution and Phylogeny*, editor: Gascuel O.. Oxford (UK): Oxford University Press. p. 33-62.

Cook R. D. 1986. Assessment of local influence (with discussion), *J. Roy. Statist. Soc. B*, 48: 133-169.

Efron B., Hinkley D.V. 1978. Assessing the accuracy of the maximum likelihood estimator: observed and expected information. Biometrika 65:457-487.

Felsenstein J. 1973. Maximum likelihood and minimum-steps methods for estimating evolutionary trees from data on discrete characters. *Systematic Zoology* 22: 240–249.

Felsenstein J. 1981. Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution.* 17: 368-376.

Felsenstein J., Churchill G. A. 1996. A hidden Markov model approach to variation among sites in rate of evolution. *Molecular Biology and Evolution.* 13:93-104.

Goldman N., Yang Z. 1994. A codon-based model of nucleotide substitution for protein-coding DNA sequences. *Molecular Biology and Evolution.* 11: 725-736.

Jayaswal V., Jermiin L. S., Poladian L., Robinson J. 2010. Two stationary, non-homogeneous Markov models of nucleotide sequence evolution. *Systematic Biology*, Systematic Biology Advance Access.

Jayaswal V., Jermiin L. S., Robinson J. 2005. Estimation of phylogeny using a general Markov model. *Evolutionary Bioinformatics Online*. 1:62-80.

Jennrich R. I., Bright P. B. 1976. Fitting Systems of Linear Differential Equations Using Computer Generated Exact Derivatives. *Technometrics* 18: 385-392

Kalbfleisch J. D., Lawless J. F. 1985. The analysis of panel data under a Markov assumption, *Journal of the American Statistical Association* 80: 863-871

Kishino H., Miyata T., and Hasegawa M. 1990. Maximum likelihood inference of protein phylogeny, and the origin of chloroplasts. *Journal of Molecular Evolution* 31:151¨C160.

Lanave C., Preparata G., Saccone C., Serio G. 1984. A new method for calculating evolutionary substitution rates. *Journal of Molecular Evolution* 20:86-93.

Nielsen R., Yang Z. 1998. Likelihood models for detecting positively selected amino acid sites and applications to the HIV-1 envelope gene. *Genetics*. 148: 929-936.

Porter J. 2002. Efficiency of Covariance Matrix Estimators for Maximum Likelihood Estimation. *Journal of Business & Economic Statistics*. Vol 20, No. 3: 431-440.

Schadt E. E., Lange K. 2002. Codon and Rate Variation Models in Molecular Phylogeny. *Molecular Biology and Evolution* 19: 1534-1549

Schadt E. E., Sinsheimer J. S., Lange K. 1998. Computational advances in maximum likelihood methods for molecular phylogeny. *Genome Research* 8: 222-233.

Seo T.-K., Kishino H., Thorne J. L. 2004. Estimating Absolute Rates of Synonymous and Nonsynonymous Nucleotide Substitution in Order to Characterize Natural Selection and Date Species Divergences. *Molecular Biology and Evolution* 21: 1201-1213

Thorne, J. L., Kishino H., Painter I. S., 1998. Estimating the rate of evolution of the rate of molecular evolution. *Molecular Biology and Evolution.* 15 (12): 1647-1657.

Yang Z. 1994. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: Approximate methods *Journal of Molecular Evolution* 39: 306-314

Yang Z. 1995. A space-time process model for the evolution of DNA sequences. *Genetics*. 139:993-1005.

Yang Z. 2007. PAML 4: Phylogenetic Analysis by Maximum Likelihood. *Molecular Biology and Evolution.* 24: 1586-1591.

Yang Z., Nielsen R., Goldman N., Pedersen A.-M. K. 2000. Codon-substitution models for heterogeneous selection pressure at amino acid sites. *Genetics*. 155: 431-449.