Introduction

This document illustrates some of the phangorn [4] specialised features which are useful but maybe not as well-known or just not (yet) described elsewhere. This is mainly interesting for someone who wants to explore different models or set up some simulation studies. We show how to construct data objects for different character states other than nucleotides or amino acids or how to set up different models to estimate transition rate.

The vignette Trees describes in detail how to estimate phylogenies from nucleotide or amino acids.

1 User defined data formats

To better understand how to define our own data type it is useful to know a bit more about the internal representation of phyDat objects. The internal representation of phyDat object is very similar to factor objects.

As an example we will show here several possibilities to define nucleotide data with gaps defined as a fifth state. Ignoring gaps or coding them as ambiguous sites - as it is done in most programs, also in phangorn as default - may be misleading (see Warnow(2012)[6]). When the number of gaps is low and the gaps are missing at random coding gaps as separate state may be not important.

Let assume we have given a matrix where each row contains a character vector of a taxonomical unit:

```r
> library(phangorn)
> data = matrix(c("r","a","y","g","g","a","c","-","c","t","c","g",
+ "a","a","t","g","g","a","t","-","c","t","c","a",
+ "a","a","t","-","g","a","c","c","c","t","?","g"),
+ dimnames = list(c("t1", "t2", "t3"),NULL), nrow=3, byrow=TRUE)
> data
```

*mailto:klaus.schliep@gmail.com*
Normally we would transform this matrix into an `phyDat` object and gaps are handled as ambiguous character like "?".

```r
gapsdata1 = phyDat(data)
gapsdata1
3 sequences with 12 character and 11 different site patterns.
The states are a c g t
```

Now we will define a "USER" defined object and have to supply a vector levels of the character states for the new data, in our case the for nucleotide states and the gap. Additional we can define ambiguous states which can be any of the states.

```r
gapsdata2 = phyDat(data, type="USER", levels=c("a","c","g","t","-"),
+ ambiguity = c("?", "n"))
gapsdata2
3 sequences with 10 character and 9 different site patterns.
The states are a c g t
```

This is not yet what we wanted as two sites of our alignment, which contain the ambiguous characters "r" and "y", got deleted. To define ambiguous characters like "r" and "y" explicitly we have to supply a contrast matrix similar to contrasts for factors.

```r
contrast = matrix(data = c(1,0,0,0,0,
+ 0,1,0,0,0,
+ 0,0,1,0,0,
+ 0,0,0,1,0,
+ 1,0,1,0,0,
+ 0,1,0,1,0,
+ 0,0,0,0,1,
+ 1,1,1,1,0,
+ 1,1,1,1,1),
+ ncol = 5, byrow = TRUE)
dimnames(contrast) = list(c("a","c","g","t","r","y","-"),
+ c("a", "c", "g", "t", "-"))
contrast
  a c g t -
a 1 0 0 0 0
c 0 1 0 0 0
```
```r
> gapsdata3 = phyDat(data, type="USER", contrast=contrast)
> gapsdata3

3 sequences with 12 character and 11 different site patterns.
The states are a c g t -
```

Here we defined "n" as a state which can be any nucleotide but not a gap "-" and "?" can be any state including a gap.

These data can be used in all functions available in phangorn to compute distance matrices or perform parsimony and maximum likelihood analysis.

# 2 Markov models of character evolution

To model nucleotide substitutions across the edges of a tree $T$ we can assign a transition matrix. In the case of nucleotides, with four character states, each $4 \times 4$ transition matrix has, at most, 12 free parameters.

Time-reversible Markov models are used to describe how characters change over time, and use fewer parameters. Time-reversible means that these models need not be directed in time, and the Markov property states that these models depend only on the current state. These models are used in analysis of phylogenies using maximum likelihood and MCMC, computing pairwise distances, as well in simulating sequence evolution.

We will now describe the General Time-Reversible (GTR) model [5]. The parameters of the GTR model are the equilibrium frequencies $\pi = (\pi_A, \pi_C, \pi_G, \pi_T)$ and a rate matrix $Q$ which has the form

$$
Q = \begin{pmatrix}
* & \alpha \pi_C & \beta \pi_G & \gamma \pi_T \\
\alpha \pi_A & * & \delta \pi_G & \epsilon \pi_T \\
\beta \pi_A & \delta \pi_C & * & \eta \pi_T \\
\gamma \pi_A & \epsilon \pi_C & \eta \pi_G & *
\end{pmatrix}
$$

(1)

where we need to assign 6 parameters $\alpha, \ldots, \eta$. The elements on the diagonal are chosen so that the rows sum to zero. The Jukes-Cantor (JC) [1] model can be derived as special case from the GTR model, for equal equilibrium frequencies $\pi_A = \pi_C = \pi_G = \pi_T = 0.25$ and equal rates set to $\alpha = \beta = \gamma = \delta = \eta$. Table 2 lists all built-in nucleotide models in phangorn. The transition probabilities which describe the probabilities of change from character $i$ to $j$ in time $t$, are given by the corresponding entries of the matrix
exponential

\[ P(t) = (p_{ij}(t)) = e^{Qt}, \quad \sum_j p_{ij} = 1 \]

where \( P(t) \) is the transition matrix spanning a period of time \( t \).

### 3 Estimation of non-standard transition rate matrices

In the last section we described how to set up user defined data formats. Now we describe how to estimate transition matrices with pml.

Again for nucleotide data the most common models can be called directly in the `optim.pml` function (e.g. "JC69", "HKY", "GTR" to name a few). Table 2 lists all the available nucleotide models, which can estimated directly in `optim.pml`. For amino acids several transition matrices are available ("WAG", "JTT", "LG", "Dayhoff", "cpREV", "mt-mam", "mtArt", "Mtzoa", "mtREV24", "VT", "RtREV", "HIVw", "HIVb", "FLU", "Blossum62", "Dayhoff_DCMut" and "JTT-DCMut") or can be estimated with `optim.pml`. For example Mathews et al. (2010) [2] used this function to estimate a phytochrome amino acid transition matrix.

We will now show how to estimate a rate matrix with different transition (\( \alpha \)) and transversion ratio (\( \beta \)) and a fixed rate to the gap state (\( \gamma \)) - a kind of Kimura two-parameter model (K81) for nucleotide data with gaps as fifth state (see table 1).

\[
\begin{array}{c|cccc}
  & a & c & g & t \\
 a & \beta & & & \\
 c & & \alpha & \beta & \\
 g & \beta & \alpha & \beta & \\
 t & & & & \\
 - & \gamma & \gamma & \gamma & \gamma \\
\end{array}
\]

Table 1: Rate matrix K to optimise.

If we want to fit a non-standard transition rate matrices, we have to tell `optim.pml`, which transitions in \( Q \) get the same rate. The parameter vector `subs` accepts a vector of consecutive integers and at least one element has to be zero (these gets the reference rate of 1). Negative values indicate that there is no direct transition is possible and the rate is set to zero.

```r
> library(ape)
> tree = unroot(rtree(3))
> fit = pml(tree, gapsdata3)
> fit = optim.pml(fit, optQ=TRUE, subs=c(1,0,1,2,1,0,2,1,2,2),
```
> fit
loglikelihood: -33.00773
unconstrained loglikelihood: -28.43259

Rate matrix:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>c</th>
<th>g</th>
<th>t</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.000000e+00</td>
<td>2.584351e-06</td>
<td>1.000000e+00</td>
<td>2.584351e-06</td>
<td>0.6911908</td>
</tr>
<tr>
<td>c</td>
<td>2.584351e-06</td>
<td>0.000000e+00</td>
<td>2.584351e-06</td>
<td>1.000000e+00</td>
<td>0.6911908</td>
</tr>
<tr>
<td>g</td>
<td>1.000000e+00</td>
<td>2.584351e-06</td>
<td>0.000000e+00</td>
<td>2.584351e-06</td>
<td>0.6911908</td>
</tr>
<tr>
<td>t</td>
<td>2.584351e-06</td>
<td>1.000000e+00</td>
<td>2.584351e-06</td>
<td>0.000000e+00</td>
<td>0.6911908</td>
</tr>
<tr>
<td>-</td>
<td>6.911908e-01</td>
<td>6.911908e-01</td>
<td>6.911908e-01</td>
<td>6.911908e-01</td>
<td>0.0000000</td>
</tr>
</tbody>
</table>

Base frequencies:
0.2 0.2 0.2 0.2

Here are some conventions how the models are estimated:

If a model is supplied the base frequencies bf and rate matrix Q are optimised according to the model (nucleotides) or the adequate rate matrix and frequencies are chosen (for amino acids). If optQ=TRUE and neither a model or subs are supplied than a symmetric (optBf=FALSE) or reversible model (optBf=TRUE, i.e. the GTR for nucleotides) is estimated. This can be slow if the there are many character states, e.g. for amino acids.

4 Codon substitution models

A special case of the transition rates are codon models. *phangorn* now offers the possibility to estimate the $d_N/d_S$ ratio (sometimes called $ka/ks$), for an overview see [7]. These functions extend the option to estimates the $d_N/d_S$ ratio for pairwise sequence comparison as it is available through the function *kaks* in *seqinr*. The transition rate between between codon $i$ and $j$ is defined as follows:

\[
q_{ij} = \begin{cases} 
0 & \text{if } i \text{ and } j \text{ differ in more than one position} \\
\pi_j & \text{for synonymous transversion} \\
\pi_j\kappa & \text{for synonymous transition} \\
\pi_j\omega & \text{for non-synonymous transversion} \\
\pi_j\omega\kappa & \text{for non synonymous transition}
\end{cases}
\]

where $\omega$ is the $d_N/d_S$ ratio, $\kappa$ the transition transversion ratio and $\pi_j$ is the the equilibrium frequencies of codon $j$. For $\omega \sim 1$ the an amino acid change is neutral, for
<table>
<thead>
<tr>
<th>model</th>
<th>optQ</th>
<th>optBf</th>
<th>subs</th>
<th>df</th>
</tr>
</thead>
<tbody>
<tr>
<td>JC</td>
<td>FALSE</td>
<td>FALSE</td>
<td>c(0,0,0,0,0,0)</td>
<td>0</td>
</tr>
<tr>
<td>F81</td>
<td>FALSE</td>
<td>TRUE</td>
<td>c(0,0,0,0,0,0)</td>
<td>3</td>
</tr>
<tr>
<td>K80</td>
<td>TRUE</td>
<td>FALSE</td>
<td>c(0,1,0,1,0,0)</td>
<td>1</td>
</tr>
<tr>
<td>HKY</td>
<td>TRUE</td>
<td>TRUE</td>
<td>c(0,1,0,0,1,0)</td>
<td>4</td>
</tr>
<tr>
<td>TrNe</td>
<td>TRUE</td>
<td>FALSE</td>
<td>c(0,1,0,0,2,0)</td>
<td>2</td>
</tr>
<tr>
<td>TrN</td>
<td>TRUE</td>
<td>TRUE</td>
<td>c(0,1,0,0,2,0)</td>
<td>5</td>
</tr>
<tr>
<td>TPM1</td>
<td>TRUE</td>
<td>FALSE</td>
<td>c(0,1,2,2,1,0)</td>
<td>2</td>
</tr>
<tr>
<td>K81</td>
<td>TRUE</td>
<td>FALSE</td>
<td>c(0,1,2,2,1,0)</td>
<td>2</td>
</tr>
<tr>
<td>TPM1u</td>
<td>TRUE</td>
<td>TRUE</td>
<td>c(0,1,2,2,1,0)</td>
<td>5</td>
</tr>
<tr>
<td>TPM2</td>
<td>TRUE</td>
<td>FALSE</td>
<td>c(1,2,1,0,2,0)</td>
<td>2</td>
</tr>
<tr>
<td>TPM2u</td>
<td>TRUE</td>
<td>TRUE</td>
<td>c(1,2,1,0,2,0)</td>
<td>5</td>
</tr>
<tr>
<td>TPM3</td>
<td>TRUE</td>
<td>FALSE</td>
<td>c(1,2,0,1,2,0)</td>
<td>2</td>
</tr>
<tr>
<td>TPM3u</td>
<td>TRUE</td>
<td>TRUE</td>
<td>c(1,2,0,1,2,0)</td>
<td>5</td>
</tr>
<tr>
<td>TIM1e</td>
<td>TRUE</td>
<td>FALSE</td>
<td>c(0,1,2,2,3,0)</td>
<td>3</td>
</tr>
<tr>
<td>TIM1</td>
<td>TRUE</td>
<td>TRUE</td>
<td>c(0,1,2,2,3,0)</td>
<td>6</td>
</tr>
<tr>
<td>TIM2e</td>
<td>TRUE</td>
<td>FALSE</td>
<td>c(1,2,1,0,3,0)</td>
<td>3</td>
</tr>
<tr>
<td>TIM2</td>
<td>TRUE</td>
<td>TRUE</td>
<td>c(1,2,1,0,3,0)</td>
<td>6</td>
</tr>
<tr>
<td>TIM3e</td>
<td>TRUE</td>
<td>FALSE</td>
<td>c(1,2,0,1,3,0)</td>
<td>3</td>
</tr>
<tr>
<td>TIM3</td>
<td>TRUE</td>
<td>TRUE</td>
<td>c(1,2,0,1,3,0)</td>
<td>6</td>
</tr>
<tr>
<td>TVMe</td>
<td>TRUE</td>
<td>FALSE</td>
<td>c(1,2,3,4,2,0)</td>
<td>4</td>
</tr>
<tr>
<td>TVM</td>
<td>TRUE</td>
<td>TRUE</td>
<td>c(1,2,3,4,2,0)</td>
<td>7</td>
</tr>
<tr>
<td>SYM</td>
<td>TRUE</td>
<td>FALSE</td>
<td>c(1,2,3,4,5,0)</td>
<td>5</td>
</tr>
<tr>
<td>GTR</td>
<td>TRUE</td>
<td>TRUE</td>
<td>c(1,2,3,4,5,0)</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 2: DNA models available in phangorn, how they are defined and number of parameters to estimate. The elements of the vector subs correspond to $\alpha, \ldots, \eta$ in equation (1).
\( \omega < 1 \) purifying selection and \( \omega > 1 \) positive selection. There are four models available: “codon0”, where both parameters \( \kappa \) and \( \omega \) are fixed to 1, “codon1” where both parameters are estimated and “codon2” or “codon3” where \( \kappa \) or \( \omega \) is fixed to 1.

We compute the \( d_N/d_S \) for some sequences given a tree using the ML functions `pml` and `optim.pml`. First we have to transform the nucleotide sequences into codons (so far the algorithms always takes triplets).

```r
> library(phangorn)
> fdir <- system.file("extdata/trees", package = "phangorn")
> primates <- read.phyDat(file.path(fdir, "primates.dna"),
+   format = "interleaved")
> tree <- NJ(dist.ml(primates))
> dat <- dna2codon(primates)
> fit <- pml(tree, dat, bf="F3x4")
> fit0 <- optim.pml(fit, model="codon0", control=pml.control(trace=0))
> fit1 <- optim.pml(fit, model="codon1", control=pml.control(trace=0))
> fit2 <- optim.pml(fit, model="codon2", control=pml.control(trace=0))
> fit3 <- optim.pml(fit, model="codon3", control=pml.control(trace=0))
> anova(fit0, fit2, fit3, fit1)
```

| Log lik. Df Df change Diff log lik. Pr(>|Chi|) |
|-------------|---------|-----------------|-----------------|-----------------|
| 1            | -2867.0 | 34              |                      |                |
| 2            | -2865.9 | 35              | 1                | 2.14            | 0.1440          |
| 3            | -2526.0 | 35              | 0                | 679.92          | <2e-16          |
| 4            | -2525.2 | 36              | 1                | 1.64            | 0.2002          |

There are several ways to estimate the codon frequencies \( \pi_j \). The simplest model is to assume they have equal frequencies (\( =1/61 \)). A second is to use the empirical codon frequencies, either computed using `baseFreq` or using the argument `bf="empirical"` in `pml`. Last but not least the frequencies can be derived from the base frequencies at each codon position, the F3x4 model is set by the argument `bf="F3x4"`. One can estimate the codon frequencies setting the option to `optBf=TRUE` in `optim.pml`. As the convergence of the 60 parameters the convergence is likely slow set the maximal iterations to a higher value than the default (e.g. `control = pml.control(maxit=50)`). Similar parameters of the F3x4 can also estimated using ML `optF3x4=TRUE` instead.

The “YN98” model [?] is similar to the ”codon1”, but addtional optimises the codon frequencies.

## 5 Generating trees

`phangorn` has several functions to generate tree topologies, which may are interesting for simulation studies. `allTrees` computes all possible bifurcating tree topologies either
rooted or unrooted for up to 10 taxa. One has to keep in mind that the number of trees is growing exponentially, use \texttt{(howmanytrees)} from \textit{ape} as a reminder.

\begin{verbatim}
> trees <- allTrees(5)
> par(mfrow=c(3,5), mar=rep(0,4))
> for(i in 1:15) plot(trees[[i]], cex=1, type="u")
\end{verbatim}

\texttt{nni} returns a list of all trees which are one nearest neighbor interchange away.

\begin{verbatim}
> trees = nni(trees[[1]])
\end{verbatim}

\texttt{rNNI} and \texttt{rSPR} generate trees which are a defined number of NNI (nearest neighbor interchange) or SPR (subtree pruning and regrafting) away.
References


6 Session Information

The version number of R and packages loaded for generating the vignette were:

- R version 3.6.0 (2019-04-26), x86_64-pc-linux-gnu
- Locale: LC_CTYPE=en_US.UTF-8, LC_NUMERIC=C, LC_TIME=en_US.UTF-8, LC_COLLATE=C, LC_MONETARY=en_US.UTF-8, LC_MESSAGES=en_US.UTF-8, LC_PAPER=en_US.UTF-8, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8, LC_IDENTIFICATION=C
- Running under: Ubuntu 18.04.2 LTS
- Matrix products: default
- BLAS: /usr/lib/x86_64-linux-gnu/openblas/libblas.so.3
- LAPACK: /usr/lib/x86_64-linux-gnu/libopenblas-r0.2.20.so
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: ape 5.3, phangorn 2.5.5, xtable 1.8-4
• Loaded via a namespace (and not attached): Matrix 1.2-17, Rcpp 1.0.1, compiler 3.6.0, fastmatch 1.1-0, grid 3.6.0, igraph 1.2.4.1, knitr 1.23, lattice 0.20-38, magrittr 1.5, nlme 3.1-140, parallel 3.6.0, pkgconfig 2.0.2, quadprog 1.5-7, tools 3.6.0, xfun 0.7