Special features of phangorn (Version 2.0.0)

Klaus P. Schliep*

December 9, 2015

Introduction

This document illustrates some of the *phangorn* [3] 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)[4]). 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:

```
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

```
   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
t1 "r"  "a"  "y"  "g"  "g"  "a"  "c"  "-"  "c"  "t"   "c"   "g"
t2 "a"  "a"  "t"  "g"  "g"  "a"  "t"  "-"  "c"  "t"   "c"   "a"
t3 "a"  "a"  "t"  "-"  "g"  "a"  "c"  "c"  "c"  "t"   "?"   "g"
```

Normally we would transform this matrix into an phyDat object and gaps are handled as ambiguous character like "?".

```
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.

```
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.

```
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","-","n","?"),
     c("a", "c", "g", "t", "-"))
contrast
  a c g t -
a 1 0 0 0 0
c 0 1 0 0 0
```

```
g 0 0 1 0 0
t 0 0 0 1 0
r 1 0 1 0 0
y 0 1 0 1 0
- 0 0 0 0 1
n 1 1 1 1 0
? 1 1 1 1 1
```

```
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 Estimation of non-standard transition rate matrices

In the last section 1 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", "mtmam", "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) [1] used this function to estimate a phytochrome amino acid transition matrix.

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

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.

```
tree = unroot(rtree(3))
fit = pml(tree, gapsdata3)
```

	a	c	g	t	-
a					
c	β				
g	α	β			
t	β	α	β		
-	γ	γ	γ	γ	

Table 1: Rate matrix K to optimise.

```
 fit = optim.pml(fit, optQ=TRUE, subs=c(1,0,1,2,1,0,2,1,2,2),
      control=pml.control(trace=0))
 fit
 loglikelihood: -33.00773

unconstrained loglikelihood: -28.43259

Rate matrix:
             a            c            g            t         -
a 0.000000e+00 2.584351e-06 1.000000e+00 2.584351e-06 0.6911908
c 2.584351e-06 0.000000e+00 2.584351e-06 1.000000e+00 0.6911908
g 1.000000e+00 2.584351e-06 0.000000e+00 2.584351e-06 0.6911908
t 2.584351e-06 1.000000e+00 2.584351e-06 0.000000e+00 0.6911908
- 6.911908e-01 6.911908e-01 6.911908e-01 6.911908e-01 0.0000000

Base frequencies:
0.2 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.

3 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 [5]. These functions extend the option to estimates the d_N/d_S ratio for pairwise sequence

model	optQ	optBf	subs	df
JC	FALSE	FALSE	$c(0,0,0,0,0,0)$	0
F81	FALSE	TRUE	$c(0,0,0,0,0,0)$	3
K80	TRUE	FALSE	$c(0,1,0,0,1,0)$	1
HKY	TRUE	TRUE	$c(0,1,0,0,1,0)$	4
TrNe	TRUE	FALSE	$c(0,1,0,0,2,0)$	2
TrN	TRUE	TRUE	$c(0,1,0,0,2,0)$	5
TPM1	TRUE	FALSE	$c(0,1,2,2,1,0)$	2
K81	TRUE	FALSE	$c(0,1,2,2,1,0)$	2
TPM1u	TRUE	TRUE	$c(0,1,2,2,1,0)$	5
TPM2	TRUE	FALSE	$c(1,2,1,0,2,0)$	2
TPM2u	TRUE	TRUE	$c(1,2,1,0,2,0)$	5
TPM3	TRUE	FALSE	$c(1,2,0,1,2,0)$	2
TPM3u	TRUE	TRUE	$c(1,2,0,1,2,0)$	5
TIM1e	TRUE	FALSE	$c(0,1,2,2,3,0)$	3
TIM1	TRUE	TRUE	$c(0,1,2,2,3,0)$	6
TIM2e	TRUE	FALSE	$c(1,2,1,0,3,0)$	3
TIM2	TRUE	TRUE	$c(1,2,1,0,3,0)$	6
TIM3e	TRUE	FALSE	$c(1,2,0,1,3,0)$	3
TIM3	TRUE	TRUE	$c(1,2,0,1,3,0)$	6
TVMe	TRUE	FALSE	$c(1,2,3,4,2,0)$	4
TVM	TRUE	TRUE	$c(1,2,3,4,2,0)$	7
SYM	TRUE	FALSE	$c(1,2,3,4,5,0)$	5
GTR	TRUE	TRUE	$c(1,2,3,4,5,0)$	8

Table 2: DNA models available in phangorn, how they are defined and number of parameters to estimate.

comparison as it is available through the function `kaks` in *seqinr*. The transition rate between between codon i and j is defined as follows:

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

where ω is the d_N/d_S ratio, κ the transition transversion ratio and π_j is the the equilibrium frequencies of codon j. For $\omega \sim 1$ the an amino acid change is neutral, for $\omega < 1$ purifying selection and $\omega > 1$ positive selection. There are four models available: "codon0", where both parameter κ and ω are fixed to 1, "codon1" where both parameters are estimated and "codon2" or "codon3" where κ or ω 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 the nucleotide sequences into codons (so far the algorithms always takes triplets).

```
library(phangorn)
primates = read.phyDat("primates.dna", format="phylip", type="DNA")
tree <- NJ(dist.ml(primates))
dat <- phyDat(as.character(primates), "CODON")
fit <- pml(tree, dat)
fit0 <- optim.pml(fit, 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)
Likelihood Ratio Test Table
  Log lik. Df Df change Diff log lik. Pr(>|Chi|)
1  -2905.6 25
2  -2385.7 26         1       1039.65     <2e-16
3  -2292.6 26         0        186.27     <2e-16
4  -2291.4 27         1          2.39     0.1218
```

The models described here all assume equal frequencies for each codon (=1/61). One can optimise the codon frequencies setting the option to optBf=TRUE. As the convergence of the 61 parameters the convergence is likely slow set the maximal iterations to a higher value than the default (e.g. control = pml.control(maxit=50)).

4 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

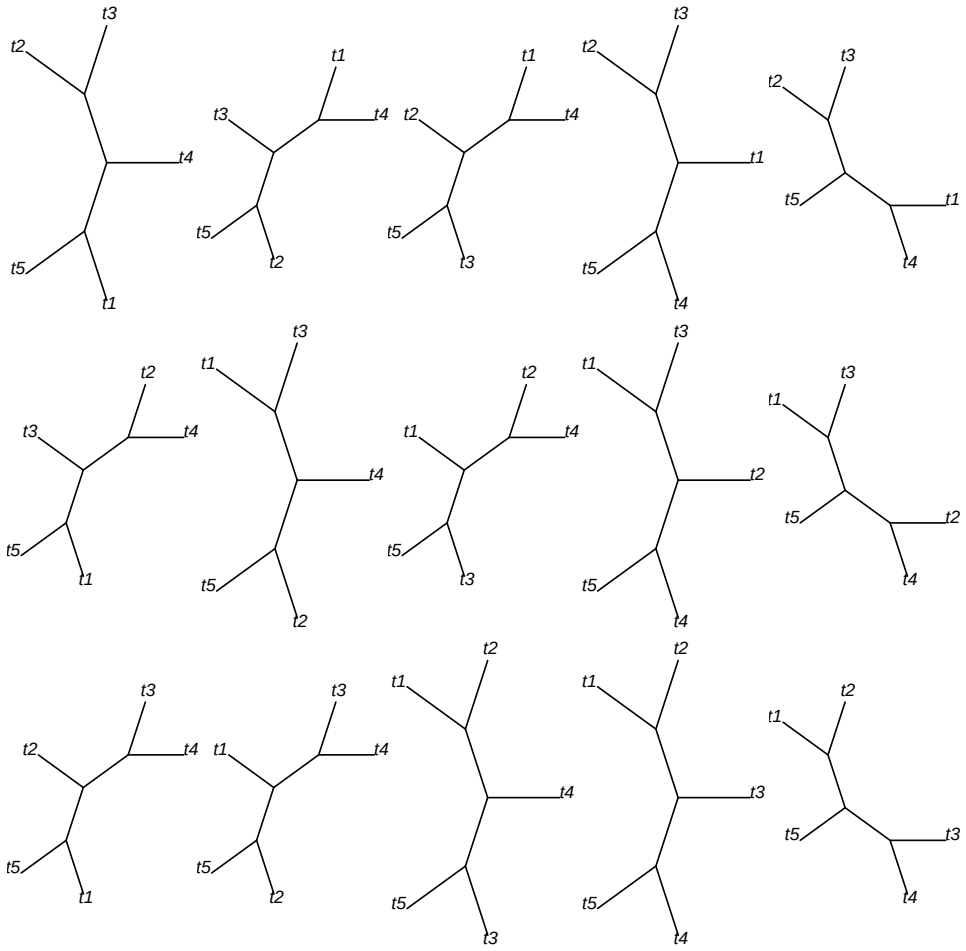

Figure 1: all (15) unrooted trees with 5 taxa

rooted or unrooted for up to 10 taxa. One has to keep in mind that the number of trees is growing exponentially, use (howmanytrees) from *ape* as a reminder.

```
trees = allTrees(5)
par(mfrow=c(3,5), mar=rep(0,4))
for(i in 1:15)plot(trees[[i]], cex=1, type="u")
```

`nni` returns a list of all trees which are one nearest neighbor interchange away.

```
trees = nni(trees[[1]])
```

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

References

[1] S. Mathews, M.D. Clements, and M.A. Beilstein. A duplicate gene rooting of seed plants and the phylogenetic position of flowering plants. *Phil. Trans. R. Soc. B*, 365:383–395, 2010.

[2] Emmanuel Paradis. *Analysis of Phylogenetics and Evolution with R*. Springer, New York, second edition, 2012.

[3] Klaus Peter Schliep. phangorn: Phylogenetic analysis in R. *Bioinformatics*, 27(4):592–593, 2011.

[4] Tandy Warnow. Standard maximum likelihood analyses of alignments with gaps can be statistically inconsistent. *PLOS Currents Tree of Life*, 2012.

[5] Ziheng Yang. *Computational Molecular evolution*. Oxford University Press, Oxford, 2006.

5 Session Information

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

- R version 3.2.2 (2015-08-14), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=en_US.UTF-8`, `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`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils
- Other packages: ape 3.4, devtools 1.9.1, phangorn 2.0.0, seqLogo 1.36.0, xtable 1.8-0
- Loaded via a namespace (and not attached): BiocGenerics 0.16.1, Biostrings 2.38.2, digest 0.6.8, evaluate 0.8, formatR 1.2.1, htmltools 0.2.6, igraph 1.0.1, IRanges 2.4.4, knitr 1.11, lattice 0.20-33, magrittr 1.5, Matrix 1.2-3, memoise 0.2.1, nlme 3.1-122, nnls 1.4, parallel 3.2.2, quadprog 1.5-5, rmarkdown 0.8.1, S4Vectors 0.8.3, stats4 3.2.2, stringi 1.0-1, stringr 1.0.0, tools 3.2.2, XVector 0.10.0, yaml 2.1.13, zlibbioc 1.16.0