Estimating phylogenetic trees with phangorn

Klaus P. Schliep∗

May 20, 2014

1 Introduction

These notes should enable the user to estimate phylogenetic trees from alignment data with different methods using the *phangorn* package [10]. Several functions of *phangorn* are also described in more detail in [6]. For more theoretical background on all the methods see e.g. [2, 12]. This document illustrates some of the *phangorn* features to estimate phylogenetic trees using different reconstruction methods. Small adaptations to the scripts in section 6 should enable the user to perform phylogenetic analyses.

2 Getting started

The first thing we have to do is to read in an alignment. Unfortunately there exists many different file formats that alignments can be stored in. The function `read.phyDat` is used to read in an alignment. There are several functions to read in alignments depending on the format of the dataset (nexus, phylip, fasta) and the kind of data (amino acid or nucleotides) in the *ape* package [5] and *phangorn*. The function `read.phyDat` calls these other functions. For the specific parameter settings available look in the help files of the function `read.dna` (for phylip, fasta, clustal format), `read.nexus.data` for nexus files. For amino acid data additional `read.aa` is called. We start our analysis loading the *phangorn* package and then reading in an alignment.

∗mailto:klaus.schliep@gmail.com
> library(phangorn)
> primates = read.phyDat("primates.dna", format="phylip", type="DNA")

### 3 Distance based methods

After reading in the alignment we can build a first tree with distance based methods. The function `dist.dna` from the ape package computes distances for many DNA substitution models. To use the function `dist.dna` we have to transform the data to class DNAbin. For amino acids the function `dist.ml` offers common substitution models ("WAG", "JTT", "LG", "Dayhoff", "cpREV", "mtmam", "mtArt", "MtZoa" and "mtREV24").

After constructing a distance matrix we reconstruct a rooted tree with UPGMA and alternatively an unrooted tree using Neighbor Joining [9][11].

```r
> dm = dist.dna(as.DNAbin(primates))
> treeUPGMA = upgma(dm)
> treeNJ = NJ(dm)
```

We can plot the trees `treeUPGMA` and `treeNJ` (figure[1]) with the commands:

```r
> layout(matrix(c(1,2), 2, 1), height=c(1,2))
> par(mar = c(.1,.1,.1,.1))
> plot(treeUPGMA, main="UPGMA")
> plot(treeNJ, "unrooted", main="NJ")
```

Distance based methods are very fast and we will use the UPGMA and NJ tree as starting trees for the maximum parsimony and maximum likelihood analyses.

### 4 Parsimony

The function `parsimony` returns the parsimony score, that is the number of changes which are at least necessary to describe the data for a given tree. We can compare the parsimony score or the two trees we computed so far:

```r
> parsimony(treeUPGMA, primates)
[1] 751
> parsimony(treeNJ, primates)
[1] 746
```
Figure 1: Rooted UPGMA tree and unrooted NJ tree
The function optim.parsimony performs tree rearrangements to find trees with a lower parsimony score. So far the only tree rearrangement implemented is nearest-neighbor interchanges (NNI). However, there is also a version of the parsimony ratchet [4] implemented, which is likely to find better trees than just doing NNI rearrangements.

```r
> treePars = optim.parsimony(treeUPGMA, primates)
Final p-score 746 after 1 nni operations
> treeRatchet = pratchet(primates, trace = 0)
> parsimony(c(treePars, treeRatchet), primates)
[1] 746 746
```

For small datasets it is also possible to find all most parsimonious trees using a branch and bound algorithm [3]. For datasets with more than 10 taxa this can take a long time and depends strongly on how tree like the data are.

```r
> (trees <- bab(subset(primates,1:10)))
```

### 5 Maximum likelihood

The last method we will describe in this vignette is Maximum Likelihood (ML) as introduced by Felsenstein [1]. We can easily compute the likelihood for a tree given the data.

```r
> fit = pml(treeNJ, data=primates)
> fit
loglikelihood: -3077.846

unconstrained loglikelihood: -1230.335

Rate matrix:
   a  c  g  t
a 0  1  1  1
C 1  0  1  1
g 1  1  0  1
t 1  1  1  0

Base frequencies:
0.25 0.25 0.25 0.25
```

The function pml returns an object of class pml. This object contains the data, the tree and many different parameters of the model like the likelihood.
etc. There are many generic functions for the class pml available, which allow
the handling of these objects.

```r
> methods(class="pml")
[1] anova.pml logLik.pml plot.pml print.pml update.pml vcov.pml
```

The object fit just estimated the likelihood for the tree it got supplied, but
the branch length are not optimized for the Jukes-Cantor model yet, which
can be done with the function optim.pml.

```r
> fitJC = optim.pml(fit, TRUE)
> logLik(fitJC)
```

With the default values pml will estimate a Jukes-Cantor model. The func-
tion update.pml allows to change parameters. We will change the model to
the GTR + Γ(4) + I model and then optimize all the parameters.

```r
> fitGTR = update(fit, k=4, inv=0.2)
> fitGTR = optim.pml(fitGTR, TRUE, TRUE, TRUE, TRUE, TRUE,
+ control = pml.control(trace = 0))
> fitGTR

loglikelihood: -2609.586

unconstrained loglikelihood: -1230.335
Proportion of invariant sites: 0.006033461
Discrete gamma model
Number of rate categories: 4
Shape parameter: 3.173311

Rate matrix:

```
   a     c     g     t
a 0.0000000 0.6470562 33.6373088 0.4058551
b 0.6470562 0.0000000 0.008346623 14.3905981
g 33.6373088 0.008346623 0.000000000 1.0000000
t 0.4058551 14.390598108 1.000000000 0.0000000
```

Base frequencies:
```
0.3919454 0.3794754 0.0402688 0.1883103
```

We can compare the objects for the JC and GTR + Γ(4) + I model using
likelihood ratio statistic

```r
> anova(fitJC, fitGTR)
```
Likelihood Ratio Test Table

Log lik. Df Df change Diff log lik. Pr(>|Chi|)
1  -3068.3 25
2  -2609.6 35 10 917.42 < 2.2e-16 ***

---
Signif. codes:  0 aã¥***aãž 0.001 aã¥**aãž 0.01 aã¥*aãž 0.05 aã¥.aãž 0.1 aã¥ aãž 1

with the AIC

> AIC(fitGTR)
[1] 5289.171
> AIC(fitJC)
[1] 6186.59

or the Shimodaira-Hasegawa test.

> SH.test(fitGTR, fitJC)

<table>
<thead>
<tr>
<th>Trees</th>
<th>ln L Diff ln L</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,]</td>
<td>-2609.586</td>
<td>0.0000</td>
</tr>
<tr>
<td>[2,]</td>
<td>-3068.295</td>
<td>458.7094</td>
</tr>
</tbody>
</table>

An alternative is to use the function modelTest to compare different models
the AIC or BIC, similar to popular program of [7, 8].

> mt = modelTest(primates)
The results of is illustrated in table [1]

The thresholds for the optimisation in modelTest are not as strict as for optim.pml and no tree rearrangements are performed. As modelTest computes and optimises a lot of models it would be a waste of computer time not to save these results. The results are saved as call together with the optimised trees in an environment and this call can be evaluated to get a “pml” object back to use for further optimisation or analysis.

> env <- attr(mt, "env")
> ls(envir=env)

[1] "data" "F81" "F81+G" "F81+G+I"
[5] "F81+I" "GTR" "GTR+G" "GTR+G+I"
[9] "GTR+I" "HKY" "HKY+G" "HKY+G+I"
[13] "HKY+I" "JC" "JC+G" "JC+G+I"
[17] "JC+I" "K80" "K80+G" "K80+G+I"
<table>
<thead>
<tr>
<th>Model</th>
<th>df</th>
<th>logLik</th>
<th>AIC</th>
<th>BIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 JC</td>
<td>25</td>
<td>-3068.42</td>
<td>6186.83</td>
<td>6273.00</td>
</tr>
<tr>
<td>2 JC+I</td>
<td>26</td>
<td>-3062.63</td>
<td>6177.26</td>
<td>6266.87</td>
</tr>
<tr>
<td>3 JC+G</td>
<td>26</td>
<td>-3066.92</td>
<td>6185.83</td>
<td>6275.45</td>
</tr>
<tr>
<td>4 JC+G+I</td>
<td>27</td>
<td>-3062.71</td>
<td>6179.43</td>
<td>6272.49</td>
</tr>
<tr>
<td>5 F81</td>
<td>28</td>
<td>-2918.17</td>
<td>5892.33</td>
<td>5988.84</td>
</tr>
<tr>
<td>6 F81+I</td>
<td>29</td>
<td>-2909.12</td>
<td>5876.24</td>
<td>5976.20</td>
</tr>
<tr>
<td>7 F81+G</td>
<td>29</td>
<td>-2912.58</td>
<td>5883.17</td>
<td>5983.12</td>
</tr>
<tr>
<td>8 F81+G+I</td>
<td>30</td>
<td>-2908.52</td>
<td>5877.04</td>
<td>5980.44</td>
</tr>
<tr>
<td>9 K80</td>
<td>26</td>
<td>-2952.94</td>
<td>5957.89</td>
<td>6047.50</td>
</tr>
<tr>
<td>10 K80+I</td>
<td>27</td>
<td>-2944.51</td>
<td>5943.02</td>
<td>6036.08</td>
</tr>
<tr>
<td>11 K80+G</td>
<td>27</td>
<td>-2944.99</td>
<td>5943.99</td>
<td>6037.05</td>
</tr>
<tr>
<td>12 K80+G+I</td>
<td>28</td>
<td>-2942.38</td>
<td>5940.76</td>
<td>6037.27</td>
</tr>
<tr>
<td>13 HKY</td>
<td>29</td>
<td>-2647.74</td>
<td>5353.48</td>
<td>5453.43</td>
</tr>
<tr>
<td>14 HKY+I</td>
<td>30</td>
<td>-2629.83</td>
<td>5319.67</td>
<td>5423.07</td>
</tr>
<tr>
<td>15 HKY+G</td>
<td>30</td>
<td>-2618.49</td>
<td>5296.99</td>
<td>5400.39</td>
</tr>
<tr>
<td>16 HKY+G+I</td>
<td>31</td>
<td>-2615.15</td>
<td>5292.30</td>
<td>5399.15</td>
</tr>
<tr>
<td>17 SYM</td>
<td>30</td>
<td>-2813.91</td>
<td>5687.83</td>
<td>5791.23</td>
</tr>
<tr>
<td>18 SYM+I</td>
<td>31</td>
<td>-2811.73</td>
<td>5685.45</td>
<td>5792.30</td>
</tr>
<tr>
<td>19 SYM+G</td>
<td>31</td>
<td>-2804.76</td>
<td>5671.53</td>
<td>5778.38</td>
</tr>
<tr>
<td>20 SYM+G+I</td>
<td>32</td>
<td>-2804.68</td>
<td>5673.36</td>
<td>5783.65</td>
</tr>
<tr>
<td>21 GTR</td>
<td>33</td>
<td>-2642.89</td>
<td>5351.78</td>
<td>5465.52</td>
</tr>
<tr>
<td>22 GTR+I</td>
<td>34</td>
<td>-2624.07</td>
<td>5316.15</td>
<td>5433.34</td>
</tr>
<tr>
<td>23 GTR+G</td>
<td>34</td>
<td>-2613.65</td>
<td>5295.30</td>
<td>5412.49</td>
</tr>
<tr>
<td>24 GTR+G+I</td>
<td>35</td>
<td>-2610.31</td>
<td>5290.62</td>
<td>5411.26</td>
</tr>
</tbody>
</table>

Table 1: Summary table of modelTest

```r
[21] "K80+I"   "SYM"   "SYM+G"   "SYM+G+I"
[25] "SYM+I"   "tree_F81" "tree_F81+G" "tree_F81+G+I"
[29] "tree_F81+I" "tree_GTR" "tree_GTR+G" "tree_GTR+G+I"
[33] "tree_GTR+I" "tree_HKY" "tree_HKY+G" "tree_HKY+G+I"
[37] "tree_HKY+I" "tree_JC" "tree_JC+G" "tree_JC+G+I"
[41] "tree_JC+I" "tree_K80" "tree_K80+G" "tree_K80+G+I"
[45] "tree_K80+I" "tree_SYM" "tree_SYM+G" "tree_SYM+G+I"
[49] "tree_SYM+I"
```

```r
> (fit <- eval(get("HKY+G+I", env), env))

```

loglikelihood: -2615.149

unconstrained loglikelihood: -1230.335

Proportion of invariant sites: 0.003869274

Discrete gamma model
Number of rate categories: 4
Shape parameter: 2.911518

Rate matrix:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>c</th>
<th>g</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.00000</td>
<td>1.00000</td>
<td>33.58626</td>
<td>1.00000</td>
</tr>
<tr>
<td>c</td>
<td>1.00000</td>
<td>0.00000</td>
<td>1.00000</td>
<td>33.58626</td>
</tr>
<tr>
<td>g</td>
<td>33.58626</td>
<td>1.00000</td>
<td>0.00000</td>
<td>1.00000</td>
</tr>
<tr>
<td>t</td>
<td>1.00000</td>
<td>33.58626</td>
<td>1.00000</td>
<td>0.00000</td>
</tr>
</tbody>
</table>

Base frequencies:
0.4129084 0.3650499 0.04424032 0.1778014

At last we may want to apply bootstrap to test how well the edges of the tree are supported:

```r
> bs = bootstrap.pml(fitJC, bs=100, optNni=TRUE, 
+ control = pml.control(trace = 0))
```

Now we can plot the tree with the bootstrap support values on the edges

```r
> par(mar=c(.1,.1,.1,.1))
> plotBS(fitJC$tree, bs)
```

Several analyses, e.g. bootstrap and modelTest, can be computationally demanding, but as nowadays most computers have several cores one can distribute the computations using the multicore package. However it is only possible to use this approach if R is running from command line ("X11"), but not using a GUI (for example "Aqua" on Macs) and unfortunately the multicore package does not work at all under Windows.

6 Appendix: Standard scripts for nucleotide or amino acid analysis

Here we provide two standard scripts which can be adapted for the most common tasks. Most likely the arguments for read.phyDat have to be adapted to accommodate your file format. Both scripts assume that the multicore package, see comments above.

```r
library(parallel) # supports parallel computing
library(phangorn)
file="myfile"
```
Figure 2: Unrooted tree with bootstrap support values
dat = read.phyDat(file)
dm = dist.ml(dat)
tree = NJ(dm)
# as alternative for a starting tree:
tree <- pratchet(dat)
# 1. alternative: estimate an GTR model
fitStart = pml(tree, dat, k=4, inv=.2)
fit = optim.pml(fitStart, TRUE, TRUE, TRUE, TRUE, TRUE)
# 2. alternative: modelTest
(mt <- modelTest(dat, multicore=TRUE))
mt$Model[which.min(mt$BIC)]
# choose best model from the table, assume now GTR+G+I
env = attr(mt, "env")
fitStart = eval(get("GTR+G+I", env), env)
fitStart = eval(get(mt$Model[which.min(mt$BIC)], env), env)
fit = optim.pml(fitStart, optNni=TRUE, optGamma=TRUE, optInv=TRUE,
               model="GTR")
bs = bootstrap.pml(fit, bs=100, optNni=TRUE, multicore=TRUE)

You can specify different several models build in which you can specify,
e.g. "WAG", "JTT", "Dayhoff", "LG". Optimising the rate matrix for amino
acids is possible, but would take a long, a very long time. So make sure to
set optBf=FALSE and optQ=FALSE in the function optim.pml, which is
also the default.

library(parallel) # supports parallel computing
library(phangorn)
file="myfile"
(dat = read.phyDat(file, type = "AA")
dm = dist.ml(dat, model="JTT")
tree = NJ(dm)
(mt <- modelTest(dat, model=c("JTT", "LG", "WAG"), multicore=TRUE))
fitStart = eval(get(mt$Model[which.min(mt$BIC)], env), env)
fitNJ = pml(tree, dat, model="JTT", k=4, inv=.2)
fit = optim.pml(fitNJ, optNni=TRUE, optInv=TRUE, optGamma=TRUE)
fit
bs = bootstrap.pml(fit, bs=100, optNni=TRUE, multicore=TRUE)

References


7 Session Information

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

- R version 3.1.0 (2014-04-10), i686-pc-linux-gnu

- Locale: LC_CTYPE=en_NZ.UTF-8, LC_NUMERIC=C,
  LC_TIME=en_NZ.UTF-8, LC_COLLATE=en_NZ.UTF-8,
  LC_MONETARY=en_NZ.UTF-8, LC_MESSAGES=en_NZ.UTF-8,
  LC_PAPER=en_NZ.UTF-8, LC_NAME=C, LC_ADDRESS=C,
LC_TELEPHONE=C, LC_MEASUREMENT=en_NZ.UTF-8, LC_IDENTIFICATION=C

- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils

- Other packages: ape 3.1-1, phangorn 1.99-8, seqLogo 1.30.0, xtable 1.7-3

- Loaded via a namespace (and not attached): fastmatch 1.0-4, igraph 0.7.1, lattice 0.20-29, Matrix 1.1-3, nlme 3.1-117, tools 3.1.0