Estimating phylogenetic trees with phangorn (Version 1.7-0)

Klaus P. Schliep*

October 30, 2012

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.

```
> library(phangorn)
> primates = read.phyDat("primates.dna", format="phylip", type="DNA")
```

*mailto:klaus.schliep@gmail.com

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].

```
> 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:

```
> 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:

```
> parsimony(treeUPGMA, primates)
[1] 751
> parsimony(treeNJ, primates)
[1] 746
```

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 is also a version of the parsimony ratchet [4] implemented, which is likely to find better trees than just doing NNI rearrangements.

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

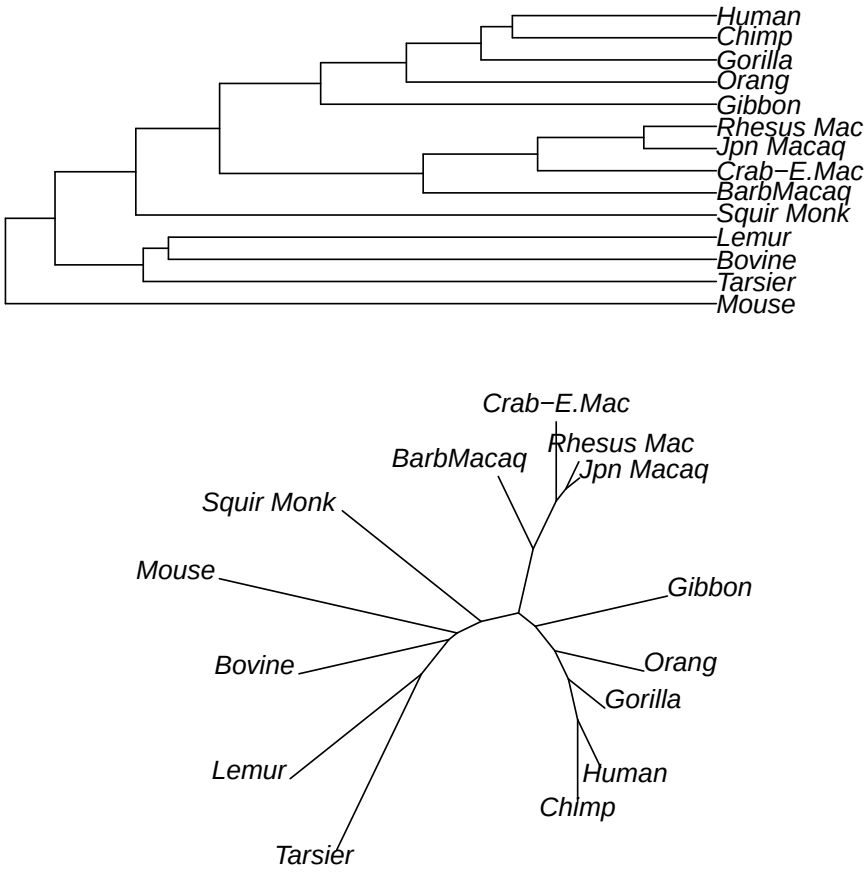

Figure 1: Rooted UPGMA tree and unrooted NJ tree

```
[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.

```
> (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

```
> 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.

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

   Non-visible functions are asterisked
```

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.

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

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

```
> fitGTR = update(fit, k=4, inv=0.2)
> fitGTR = optim.pml(fitGTR, TRUE,TRUE, TRUE, TRUE, TRUE,
+     control = pml.control(trace = 0))
> fitGTR
 loglikelihood: -2609.598

unconstrained loglikelihood: -1230.335
Proportion of invariant sites: 0.006061292
Discrete gamma model
Number of rate categories: 4
Shape parameter: 3.174819

Rate matrix:
           a            c            g          t
a  0.0000000  0.645861577 33.527747991  0.4043634
c  0.6458616  0.000000000  0.008025152 14.3385979
g 33.5277480  0.008025152  0.000000000  1.0000000
t  0.4043634 14.338597857  1.000000000  0.0000000

Base frequencies:
0.3918077 0.3795142 0.04026932 0.1884088
```

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

```
> 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        917.39  < 2.2e-16 ***
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1
```

with the AIC

```
> AIC(fitGTR)
[1] 5289.195
```

```
> AIC(fitJC)
[1] 6186.59
```

or the Shimodaira-Hasegawa test.

```
> SH.test(fitGTR, fitJC)
     Trees      ln L Diff ln L p-value
[1,]     1 -2609.598    0.0000     0.5
[2,]     2 -3068.295  458.6975     0.0
```

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"
[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"
> (fit <- eval(get("HKY+G+I", env), env))
 loglikelihood: -2615.149

unconstrained loglikelihood: -1230.335
Proportion of invariant sites: 0.003869274
```

	Model	df	logLik	AIC	BIC
1	JC	25.00	-3068.42	6186.83	6273.00
2	JC+I	26.00	-3062.63	6177.26	6266.87
3	JC+G	26.00	-3066.92	6185.83	6275.45
4	JC+G+I	27.00	-3062.71	6179.43	6272.49
5	F81	28.00	-2918.17	5892.33	5988.84
6	F81+I	29.00	-2909.12	5876.24	5976.20
7	F81+G	29.00	-2912.58	5883.17	5983.12
8	F81+G+I	30.00	-2908.52	5877.04	5980.44
9	K80	26.00	-2952.94	5957.89	6047.50
10	K80+I	27.00	-2944.51	5943.02	6036.08
11	K80+G	27.00	-2944.99	5943.99	6037.05
12	K80+G+I	28.00	-2942.38	5940.76	6037.27
13	HKY	29.00	-2647.74	5353.48	5453.43
14	HKY+I	30.00	-2629.83	5319.67	5423.07
15	HKY+G	30.00	-2618.49	5296.99	5400.39
16	HKY+G+I	31.00	-2615.15	5292.30	5399.15
17	SYM	30.00	-2813.91	5687.83	5791.23
18	SYM+I	31.00	-2811.73	5685.45	5792.30
19	SYM+G	31.00	-2804.76	5671.53	5778.38
20	SYM+G+I	32.00	-2804.68	5673.36	5783.65
21	GTR	33.00	-2642.89	5351.78	5465.52
22	GTR+I	34.00	-2624.07	5316.15	5433.34
23	GTR+G	34.00	-2613.65	5295.30	5412.49
24	GTR+G+I	35.00	-2610.31	5290.62	5411.26

Table 1: Summary table of modelTest

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

Rate matrix:
          a        c        g        t
a  0.00000  1.00000 33.58626  1.00000
c  1.00000  0.00000  1.00000 33.58626
g 33.58626  1.00000  0.00000  1.00000
t  1.00000 33.58626  1.00000  0.00000

Base frequencies:
0.4129084 0.3650499 0.04424032 0.1778014
```

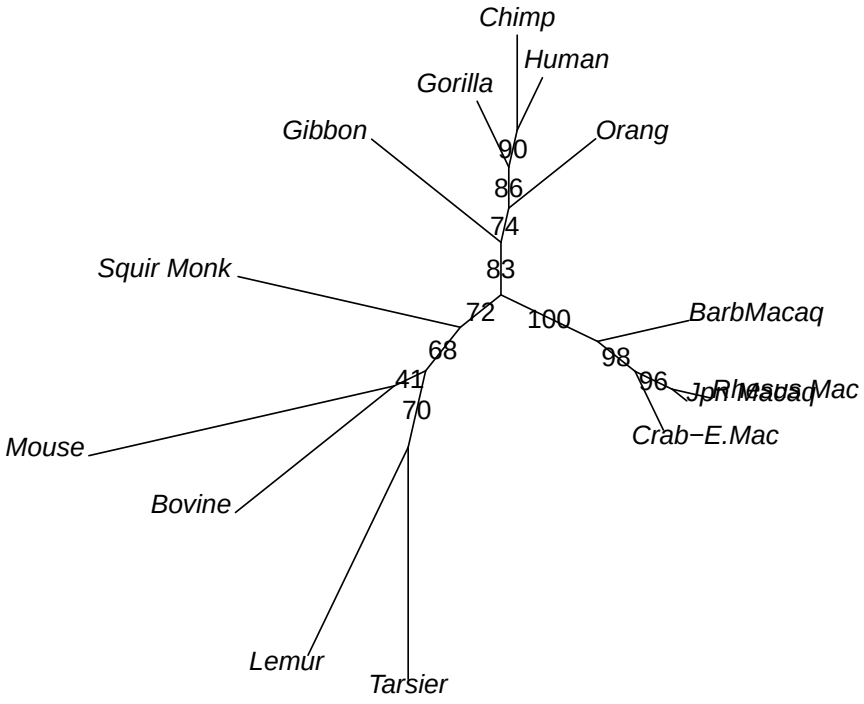

Figure 2: Unrooted tree with bootstrap support values

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

```
> 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

```
> 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.

```
library(parallel) # supports parallel computing
library(phangorn)
file="myfile"
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

[1] Joseph Felsenstein. Evolutionary trees from dna sequences: a maxumum likelihood approach. *Journal of Molecular Evolution*, 17:368–376, 1981.

[2] Joseph Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Sunderland, 2004.

[3] M.D. Hendy and Penny D. Branch and bound algorithms to determine minimal evolutionary trees. *Math. Biosc.*, 59:277–290, 1982.

[4] K.˜Nixon. The parsimony ratchet, a new method for rapid rarsimony analysis. *Cladistics*, 15:407–414, 1999.

[5] E.˜Paradis, J.˜Claude, and K.˜Strimmer. Ape: Analyses of phylogenetics and evolution in r language. *Bioinformatics*, 20(2):289–290, 2004.

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

[7] D.˜Posada and K.A. Crandall. Modeltest: testing the model of dna substitution. *Bioinformatics*, 14(9):817–818, 1998.

[8] David Posada. jmodeltest: Phylogenetic model averaging. *Molecular Biology and Evolution*, 25(7):1253–1256, 2008.

[9] N.˜Saitou and M.˜Nei. The neighbor-joining method - a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.

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

[11] J.˜A. Studier and K.˜J. Keppler. A note on the neighbor-joining algorithm of saitou and nei. *Molecular Biology and Evolution*, 5(6):729–731, 1988.

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

7 Session Information

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

- R version 2.15.2 (2012-10-26), `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=C`, `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˜3.0-6, igraph˜0.6-3, lattice˜0.20-10, Matrix˜1.0-10, phangorn˜1.7-0, seqLogo˜1.24.0, xtable˜1.7-0
- Loaded via a namespace (and not attached): gee˜4.13-18, nlme˜3.1-105, tools˜2.15.2