Estimating phylogenetic trees with phangorn

Klaus P. Schliep*

December 9, 2015

1 Introduction

These notes should enable the user to estimate phylogenetic trees from alignment data with different methods using the *phangorn* package [12]. Several functions of *phangorn* are also described in more detail in [8]. For more theoretical background on all the methods see e.g. [3, 15]. 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 data set (nexus, phylip, fasta) and the kind of data (amino acid or nucleotides) in the *ape* package [7] 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 (for example "WAG", "JTT", "LG", "Dayhoff", "cpREV", "mtmam", "mtArt", "MtZoa" or "mtREV24").

After constructing a distance matrix we reconstruct a rooted tree with UPGMA and alternatively an unrooted tree using Neighbor Joining [11, 14]. More distance methods like fastme are available in the *ape* package.

```
dm  <- dist.ml(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(0,0,2,0)+ 0.1)
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
```

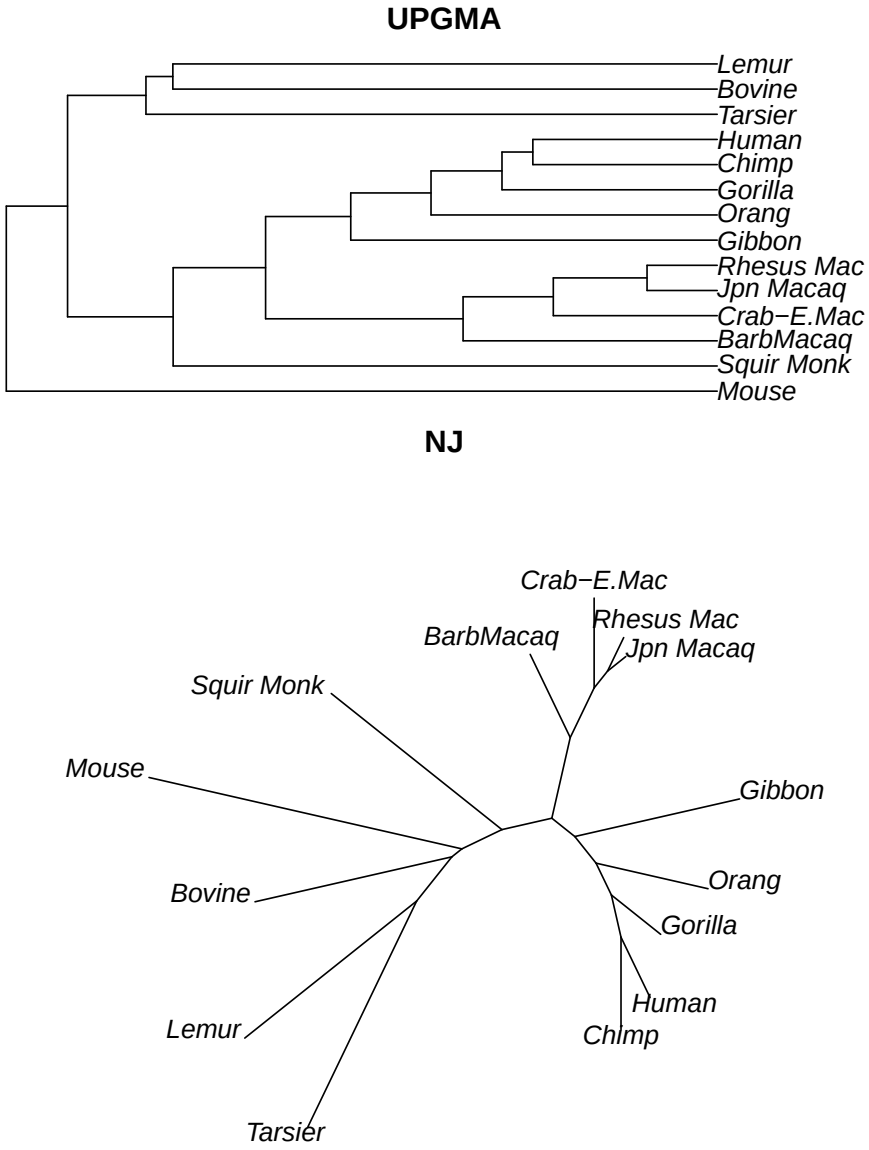

Figure 1: Rooted UPGMA tree and unrooted NJ tree

The function optim.parsimony performs tree rearrangements to find trees with a lower parsimony score. The tree rearrangement implemented are nearest-neighbor interchanges (NNI) and subtree pruning and regrafting (SPR). The later one only works so far with the fitch algorithm. However is also a version of the parsimony ratchet [6] implemented, which is likely to find better trees than just doing NNI / SPR rearrangements.

```
 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 data sets it is also possible to find all most parsimonious trees using a branch and bound algorithm [4]. For data sets 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 [2]. We can easily compute the likelihood for a tree given the data

```
 fit = pml(treeNJ, data=primates)
 fit
 loglikelihood: -3074.952

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. There are many generic functions for the class Robjectpml available, which allow the handling of these objects.

```
methods(class="pml")
```

```
[1] AICc   anova  BIC    logLik plot   print  simSeq update vcov
see '?methods' for accessing help and source code
```

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 + $\Gamma(4)$ + I model and then optimize all the parameters.

```
fitGTR <- update(fit, k=4, inv=0.2)
fitGTR <- optim.pml(fitGTR, model="GTR", optInv=TRUE, optGamma=TRUE,
     rearrangement = "NNI", control = pml.control(trace = 0))
fitGTR
```

```
loglikelihood: -2609.101

unconstrained loglikelihood: -1230.335
Proportion of invariant sites: 0.005829837
Discrete gamma model
Number of rate categories: 4
Shape parameter: 3.071244

Rate matrix:
           a            c            g          t
a  0.0000000  0.626911020 34.690620176  0.3876089
c  0.6269110  0.000000000  0.006406276 14.7063281
g 34.6906202  0.006406276  0.000000000  1.0000000
t  0.3876089 14.706328083  1.000000000  0.0000000

Base frequencies:
0.3927207 0.3793272 0.04027053 0.1876815
```

With the control parameters the threshols for the fitting process can be changed. Here we want just to supress output during the fitting process.

For larger trees the NNI rearrangements often get stuck in local maxima. We added a stochatic algortithm similar as in [5], which makes random NNI permuation to the tree, which than gets optimised to escape local optima. While this may find better trees it will also take more time.

```
fitGTR <- optim.pml(fitGTR, model="GTR", optInv=TRUE, optGamma=TRUE,
     rearrangement = "stochastic", control = pml.control(trace = 0))
fitGTR
loglikelihood: -2606.858

unconstrained loglikelihood: -1230.335
Proportion of invariant sites: 0.005584102
Discrete gamma model
Number of rate categories: 4
Shape parameter: 2.725419

Rate matrix:
           a            c            g          t
a  0.0000000  0.693501913 61.424353956  0.5380713
c  0.6935019  0.000000000  0.005264951 23.9825749
g 61.4243540  0.005264951  0.000000000  1.0000000
t  0.5380713 23.982574862  1.000000000  0.0000000

Base frequencies:
0.3962417 0.3787098 0.04029902 0.1847495
```

5.1 Model selection

We can compare nested models 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  -2606.9 35        10        922.87  < 2.2e-16
```

with the Shimodaira-Hasegawa [13] test

```
SH.test(fitGTR, fitJC)
     Trees      ln L Diff ln L p-value
[1,]     1 -2606.858    0.0000  0.5022
[2,]     2 -3068.295  461.4374  0.0000
```

or with the AIC

```
 AIC(fitJC)
[1] 6186.59
 AIC(fitGTR)
[1] 5283.715
 AICc(fitGTR)
[1] 5296.573
 BIC(fitGTR)
[1] 5404.351
```

An alternative is to use the function `modelTest` to compare different nucleotide or protein models the AIC, AICc or BIC, similar to popular programs ModelTest and ProtTest [9, 10, 1].

```
mt = modelTest(primates)
```

The results of `modelTest` is illustrated in table 1

The thresholds for the optimization in `modelTest` are not as strict as for `optim.pml` and no tree rearrangements are performed. As `modelTest` computes and optimizes 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 optimized trees in an environment and this call can be evaluated to get a "pml" object back to use for further optimization 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"
```

Model	df	logLik	AIC	AICw	AICc	AICcw	BIC
JC	25.00	-3068.42	6186.83	0.00	6193.15	0.00	6273.00
JC+I	26.00	-3062.63	6177.26	0.00	6184.10	0.00	6266.87
JC+G	26.00	-3066.92	6185.83	0.00	6192.68	0.00	6275.45
JC+G+I	27.00	-3062.64	6179.28	0.00	6186.70	0.00	6272.35
F81	28.00	-2918.17	5892.33	0.00	5900.33	0.00	5988.84
F81+I	29.00	-2909.12	5876.24	0.00	5884.85	0.00	5976.20
F81+G	29.00	-2912.56	5883.12	0.00	5891.73	0.00	5983.07
F81+G+I	30.00	-2908.52	5877.04	0.00	5886.29	0.00	5980.44
K80	26.00	-2952.94	5957.89	0.00	5964.73	0.00	6047.50
K80+I	27.00	-2944.51	5943.02	0.00	5950.43	0.00	6036.08
K80+G	27.00	-2944.76	5943.53	0.00	5950.94	0.00	6036.59
K80+G+I	28.00	-2942.34	5940.68	0.00	5948.68	0.00	6037.19
HKY	29.00	-2625.04	5308.08	0.00	5316.70	0.00	5408.04
HKY+I	30.00	-2621.27	5302.54	0.00	5311.80	0.00	5405.95
HKY+G	30.00	-2612.64	5285.28	0.18	5294.54	0.45	5388.69
HKY+G+I	31.00	-2612.45	5286.89	0.08	5296.81	0.14	5393.74
SYM	30.00	-2813.90	5687.79	0.00	5697.05	0.00	5791.19
SYM+I	31.00	-2811.73	5685.46	0.00	5695.38	0.00	5792.31
SYM+G	31.00	-2804.68	5671.36	0.00	5681.28	0.00	5778.20
SYM+G+I	32.00	-2804.67	5673.34	0.00	5683.95	0.00	5783.63
GTR	33.00	-2618.62	5303.24	0.00	5314.57	0.00	5416.98
GTR+I	34.00	-2613.58	5295.16	0.00	5307.24	0.00	5412.35
GTR+G	34.00	-2607.66	5283.33	0.47	5295.41	0.29	5400.52
GTR+G+I	35.00	-2607.21	5284.43	0.27	5297.29	0.11	5405.06

Table 1: Summary table of modelTest

```
 (fit <- eval(get("HKY+G+I", env), env))
 loglikelihood: -2612.446

unconstrained loglikelihood: -1230.335
Proportion of invariant sites: 0.00269351
Discrete gamma model
Number of rate categories: 4
Shape parameter: 2.123698

Rate matrix:
          a        c        g        t
a  0.00000  1.00000 56.02004  1.00000
c  1.00000  0.00000  1.00000 56.02004
g 56.02004  1.00000  0.00000  1.00000
```

```
t  1.00000 56.02004  1.00000  0.00000

Base frequencies:
0.4205044 0.3622272 0.0438954 0.1733729
```

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 and also look at consensusNet to identify potential conflict.

```
par(mfrow=c(2,1))
par(mar=c(1,1,3,1))
plotBS(midpoint(fitJC$tree), bs, p = 50, type="p")
title("a)")
cnet <- consensusNet(bs, p=0.2)
plot(cnet, "2D", show.edge.label=TRUE)
title("b)")
```

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.

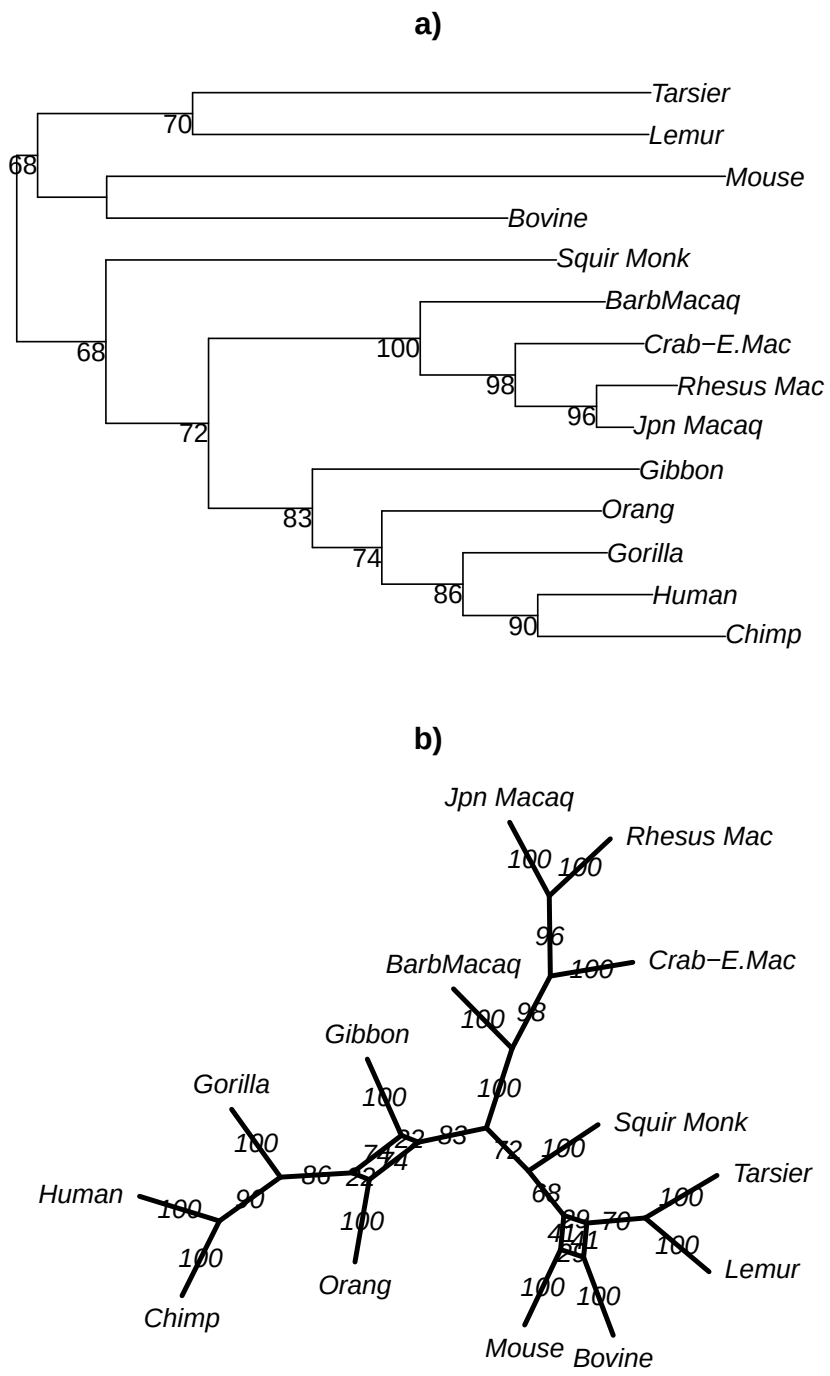

Figure 2: a) Unrooted tree (midpoint rooted) with bootstrap support values.
b) ConsensusNet from the bootstrap sample

6 Appendix: Standard scripts for nucleotide 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 works on your platform, see comments above.

```
library(phangorn)
file="myfile"
dat = read.phyDat(file)
dm = dist.ml(dat, "F81")
tree = NJ(dm)
# as alternative for a starting tree:
tree <- pratchet(dat)          # parsimony tree
tree <- nnls.phylo(tree, dm)   # need edge weights
# 1. alternative: quick and dirty: GTR + G + NNI
fitStart = pml(tree, dat, k=4)
fit = optim.pml(fitStart, model="GTR", optGamma=TRUE, rearrangement="NNI")
# 2. alternative: preper with modelTest
mt <- modelTest(dat, tree=tree, multicore=TRUE)
mt[order(mt$AICc),]
# choose best model from the table according to AICc
bestmodel <- mt$Model[which.min(mt$AICc)]
env = attr(mt, "env")
fitStart = eval(get("GTR+G+I", env), env)
# or let R search the table
fitStart = eval(get(bestmodel, env), env)
# equivalent to:   fitStart = eval(get("GTR+G+I", env), env)
fit = optim.pml(fitStart, rearrangement = "stochastic",
     optGamma=TRUE, optInv=TRUE, model="GTR")
bs = bootstrap.pml(fit, bs=100, optNni=TRUE, multicore=TRUE)
```

7 Appendix 2: Standard scripts for amino acid analysis

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

```
library(phangorn)
file="myfile"
dat = read.phyDat(file, type = "AA")
dm = dist.ml(dat, model="JTT")
tree = NJ(dm)
# parallel will only work safely from command line
# and not at all windows
(mt <- modelTest(dat, model=c("JTT", "LG", "WAG"),
     multicore=TRUE))
# run all available amino acid models
(mt <- modelTest(dat, model="all", 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, rearrangement = "stochastic",
     optInv=TRUE, optGamma=TRUE)
fit
bs = bootstrap.pml(fit, bs=100, optNni=TRUE, multicore=TRUE)
```

References

[1] Federico Abascal, Rafael Zardoya, and David Posada. Prottest: selection of best-fit models of protein evolution. *Bioinformatics*, 21(9):2104–2105, 2005.

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

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

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

[5] Lam-Tung Nguyen, Heiko A. Schmidt, Arndt von Haeseler, and Bui Quang Minh. Iq-tree: A fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies. *Molecular Biology and Evolution*, 32(1):268–274, 2015.

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

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

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

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

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

[11] 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.

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

[13] H. Shimodaira and M. Hasegawa. Multiple comparisons of log-likelihoods with applications to phylogenetic inference. *Molecular Biology and Evolution*, 16:1114–1116, 1999.

[14] 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.

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

8 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