The API to ape C code

Emmanuel Paradis, Richard Desper, Olivier Gascuel,
Vincent Lefort & Hoa Sien Cuong

March 26, 2008

Contents

1 Introduction

The present document is the application programmer interface (API) to the C code available in the ape package. The sources of ape can be found either on the CRAN (Comprehensive R Archive Network) Web site, where the current release of ape is distributed, or on ape's SVN repository where the last developments may be found. Both URIs are:

http://cran.r-project.org/src/contrib/Descriptions/ape.html
https://svn.mpl.ird.fr/ape/

More information on ape can be found at:

http://ape.mpl.ird.fr/

Only the C functions most likely to be useful to programmers are documented here. Credits and authorship may be found in the sources.

2 Data Structures

2.1 Trees

Two data stuctures are used to codes trees:

1. A recursive structure of pointers (`tree struc`) defined in 'ape/src/me.h';
2. The class `"phylo"` which is an R object and can be passed from/to C either as a whole, or as separate elements. The structure of this class is detailed in a document available on ape's web site.[1]

2.2 DNA Sequences

Nucleotides in ape are coded as `unsigned char`; this corresponds to the `"raw"` mode in R (see `?raw` in R). All details on this coding scheme, as well as on accompanying C macros, are given in a separate document.[2]

3 C functions

3.1 Trees

3.1.1 Manipulation

```
SEXP treeBuildWithTokens(SEXP nwk)
```

[1] ape.mpl.ird.fr/misc/FormatTreeR_4Dec2006.pdf
[2] ape.mpl.ird.fr/misc/BitLevelCodingScheme_20April2007.pdf

Builds a "phylo" structure from a Newich string; the latter must be tokenized (i.e., the labels must be integers) and with branch lengths.

Input	nwk: an R character vector with a single Newick string
Output	an R list with four or five elements: (i) the edge matrix (as a vector), (ii) the edge lengths, (iii) the number of nodes, (iv) the node labels (as integers), (v) optionally, the length of the root edge if present

The R function .treeBuildWithTokens sets the attributes of the "phylo" object.

```
tree *readNewickString(char *str, int numLeaves)
```

Transforms a Newick string into a tree struct.

Input	*str: a Newick string
	numLeaves: (unused?)
Output	the pointer to the tree struct

```
void tree2phylo(tree *T, int *edge1, int *edge2,
                double *el, char **tl, int n)
```

This function transforms a tree struc into a "phylo" object. It assumes the tree is unrooted and binary, so there are $2n-3$ edges. It also assumes that the labels are six-character long.

	On input	**On output**
*T	a pointer to a tree structure	*unchanged*
*edge1	an array with $2n-3$ integers	the first column of the edge matrix
*edge2	id.	the second column of the edge matrix
*el	an array with $2n-3$ doubles	the edge lengths
**tl	a double array with n strings	the tip labels
n	the number of tips	*unchanged*

```
void neworder_cladewise(int *n, int *edge1, int *edge2,
                        int *N, int *neworder)
```

Finds the order of the edges of a "phylo" object so that they are in "cladewise order" (see details in [1]).

	On input	**On output**
*n	the number of tips	*unchanged*
*edge1	the first column of the edge matrix	*unchanged*
*edge2	the second column of the edge matrix	*unchanged*
*N	the number of edges	*unchanged*
*neworder	an array with N integers	the order of the edges

```
void neworder_pruningwise(int *ntip, int *nnode, int *edge1,
          int *edge2, int *nedge, int *neworder)
```

Finds the order of the edges of a "phylo" object so that they are in a bottom-up traversal order.

	On input	**On output**
*ntip	the number of tips	*unchanged*
*nnode	the number of nodes	*unchanged*
*edge1	the first column of the edge matrix	*unchanged*
*edge2	the second column of the edge matrix	*unchanged*
*nedge	the number of edges	*unchanged*
*neworder	an array with N integers	the order of the edges

3.1.2 Estimation

```
void nj(double *D, int *N, int *edge1, int *edge2,
        double *edge_length)
```

Estimates a tree by neighbor-joining, and returns the results as a "phylo" object.

	On input	**On output**
*D	the matrix distance (lower triangle only)	*unchanged*
*N	the number of tips	*unchanged*
*edge1	an array with $2n-3$ integers	the first column of the edge matrix
*edge2	id.	the second column of the edge matrix
*edge_length	an array with $2n-3$ doubles	the edge lengths

```
void bionj(double *X, int *N, char **labels,
          int *edge1, int *edge2, double *el, char **tl)
```

Estimates a tree by BIONJ, and returns the results as a "phylo" object.

	On input	On output
`*X`	the matrix distance (lower triangle only)	*unchanged*
`*N`	the number of sequences	*unchanged*
`**labels`	the tip labels	*unchanged*
`*edge1`	an array with $2N-3$ integers	the first column of the `edge` matrix
`*edge2`	id.	the second column of the `edge` matrix
`*el`	an array with $2N-3$ doubles	the edge lengths
`**tl`	a double array with N strings	the tip labels

```
void me_b(double *X, int *N, char **labels, int *nni,
          int *edge1, int *edge2, double *el, char **tl)
void me_o(double *X, int *N, char **labels, int *nni,
          int *edge1, int *edge2, double *el, char **tl)
```

These two functions estimate a tree with the minimum evolution method using its balanced and least squares versions, respectively, and return the results as a `"phylo"` object.

	On input	On output
`*X`	the matrix distance (lower triangle only)	*unchanged*
`*N`	the number of sequences	*unchanged*
`**labels`	the tip labels	*unchanged*
`*nni`	an integer specifying to perform NNI operations	*unchanged*
`*edge1`	an array with $2N-3$ integers	the first column of the `edge` matrix
`*edge2`	id.	the second column of the `edge` matrix
`*el`	an array with $2N-3$ doubles	the edge lengths
`**tl`	a double array with N strings	the tip labels

```
void mlphylo_DNAmodel(int *n, int *s, unsigned char *SEQ,
        double *ANC, double *w, int *edge1, int *edge2,
        double *edge_length, int *npart, int *partition,
        int *model, double *xi, double *para, int *npara,
        double *alpha, int *nalpha, int *ncat,
        double *invar, int *ninvar, double *BF,
        int *search_tree, int *fixed, double *loglik)
```

This function iterates to find the MLEs of the substitution paramaters and of the branch lengths for a given tree. Though it has been completely reworked in early 2008, it is still experimental and is not further detailed here.

3.1.3 Analysis

```
SEXP seq_root2tip(SEXP edge, SEXP nbtip, SEXP nbnode)
```

This function analyses the edge matrix of a `"phylo"` object, and returns a list of vectors giving for each tip the sequence of nodes from the root to the tip.

Input	`edge`: the edge matrix of the tree (as a vector) `nbtip`: the number of tips `nbnode`: the number of nodes
Output	an R list with `nbtip` vectors of integers

```
SEXP bipartition(SEXP edge, SEXP nbtip, SEXP nbnode)
```

This function returns a list of vectors giving, for each node, the numbers of the tips descending from this node.

Input	`edge`: the edge matrix of the tree (as a vector) `nbtip`: the number of tips `nbnode`: the number of nodes
Output	an R list with `nbnode` vectors of integers

```
SEXP prop_part(SEXP TREES, SEXP nbtree, SEXP keep_partitions)
```

Analyses a list of trees, and returns the number of times each partition was observed. If `keep_partitions` is `FALSE`, only the bipartitions observed in the first tree in `TREES` are returned together with their frequencies over all trees; otherwise, all observed bipartitions are returned.

Input	`TREES`: a list of `"phylo"` objects
	`nbtree`: the number of trees in `TREES`
	`keep_partitions`: a logical indicating whether to return all bipartitions observed in `TREES`

Output	an R list of vectors of integers with an attribute `"number"` giving the frequencies of the bipartitions

3.2 DNA Sequences

3.2.1 Manipulation

```
void BaseProportion(unsigned char *x, int *n, double *BF)
```

Computes the frequencies of A, C, G, and T in a set of nucleotides. Missing or partially known nucleotides are ignored.

	On input	**On output**
`*x`	the set of DNA sequences	*unchanged*
`*n`	the number of elements in `*x`	*unchanged*
`*BF`	an array with 4 doubles	the base frequencies

```
void SegSites(unsigned char *x, int *n, int *s, int *seg)
```

Finds the segregating (variable) sites in a matrix of DNA sequences.

	On input	**On output**
`*x`	the set of DNA sequences	*unchanged*
`*n`	the number of sequences in `*x`	*unchanged*
`*s`	the number of sites in `*x`	*unchanged*
`*seg`	an array with `*s` 0's	1 for the segrating sites, 0 otherwise

```
void GlobalDeletionDNA(unsigned char *x, int *n, int *s, int *keep)
```

Finds the sites with missing (or partially known) data.

	On input	**On output**
`*x`	the set of DNA sequences	*unchanged*
`*n`	the number of sequences in `*x`	*unchanged*
`*s`	the number of sites in `*x`	*unchanged*
`*keep`	an array with `*s` 1's	1 for the sites with no missing data, 0 otherwise

3.2.2 Pairwise Distances

```
void distDNA_raw(unsigned char *x, int *n, int *s, double *d)
void distDNA_raw_pairdel(unsigned char *x, int *n, int *s, double *d)
void distDNA_JC69(unsigned char *x, int *n, int *s, double *d,
               int *variance, double *var, int *gamma, double *alpha)
void distDNA_JC69_pairdel(unsigned char *x, int *n, int *s, double *d,
               int *variance, double *var, int *gamma, double *alpha)
void distDNA_K80(unsigned char *x, int *n, int *s, double *d,
        int *variance, double *var, int *gamma, double *alpha)
void distDNA_K80_pairdel(unsigned char *x, int *n, int *s, double *d,
         int *variance, double *var, int *gamma, double *alpha)
void distDNA_F81(unsigned char *x, int *n, int *s, double *d,
    double *BF, int *variance, double *var, int *gamma, double *alpha)
void distDNA_F81_pairdel(unsigned char *x, int *n, int *s, double *d,
    double *BF, int *variance, double *var, int *gamma, double *alpha)
void distDNA_K81(unsigned char *x, int *n, int *s, double *d,
                int *variance, double *var)
void distDNA_K81_pairdel(unsigned char *x, int *n, int *s, double *d,
                         int *variance, double *var)
void distDNA_F84(unsigned char *x, int *n, int *s, double *d,
                double *BF, int *variance, double *var)
void distDNA_F84_pairdel(unsigned char *x, int *n, int *s, double *d,
                         double *BF, int *variance, double *var)
```

```
void distDNA_T92(unsigned char *x, int *n, int *s, double *d,
                 double *BF, int *variance, double *var)
void distDNA_T92_pairdel(unsigned char *x, int *n, int *s,
              double *d, double *BF, int *variance, double *var)
void distDNA_TN93(unsigned char *x, int *n, int *s, double *d,
                  double *BF, int *variance, double *var,
                  int *gamma, double *alpha)
void distDNA_TN93_pairdel(unsigned char *x, int *n, int *s,
                double *d, double *BF, int *variance,
                double *var, int *gamma, double *alpha)
void distDNA_GG95(unsigned char *x, int *n, int *s, double *d,
                  int *variance, double *var)
void distDNA_GG95_pairdel(unsigned char *x, int *n, int *s,
                double *d, int *variance, double *var)
void distDNA_LogDet(unsigned char *x, int *n, int *s, double *d,
                    int *variance, double *var)
void distDNA_LogDet_pairdel(unsigned char *x, int *n, int *s,
              double *d, int *variance, double *var)
void distDNA_BH87(unsigned char *x, int *n, int *s, double *d,
                  int *variance, double *var)
void distDNA_ParaLin(unsigned char *x, int *n, int *s,
            double *d, int *variance, double *var)
void distDNA_ParaLin_pairdel(unsigned char *x, int *n, int *s,
            double *d, int *variance, double *var)
```

These 23 functions compute pairwise evolutionary distances for a matrix of DNA sequences. The versions without `_pairdel` assumes there are no missing data. The `_pairdel` versions check for missing data for each comparison. Only the Barry–Hartigan '87 model has no `_pairdel` version because it is based on a contingency table of nucleotides for each pair of sequences.

	On input	**On output**
`*x`	the set of DNA sequences	*unchanged*
`*n`	the number of sequences in `*x`	*unchanged*
`*s`	the number of sites in `*x`	*unchanged*
`*d`	an array with $n(n-1)/2$ doubles[a]	the distances
`*variance`	an integer: 1 to compute the variances of the distances, 0 otherwise	*unchanged*
`*var`	an array with $n(n-1)/2$ doubles	the variances of the distances
`*gamma`	an integer: 1 to apply a Γ-correction on the distances, 0 otherwise	*unchanged*
`*alpha`	the value of α to use if `*gamma==1`	*unchanged*
`*BF`	the base frequencies (A, C, G, T)	*unchanged*

[a] n^2 doubles in case of the BH87 distance which is asymmetric.

```
void dist_dna(unsigned char *x, int *n, int *s, int *model, double *d,
              double *BF, int *pairdel, int *variance, double *var,
              int *gamma, double *alpha)
```

This is the main driver that calls the functions above; its arguments are the same, in addition:

	On input	**On output**
`*model`	an integer specifying the model	*unchanged*
`*pairdel`	an integer: 1 to perform pairwise deletion, 0 otherwise	*unchanged*

3.2.3 Transition Probabilities

```
void PMAT_JC69(double t, double u, double *P)
void PMAT_K80(double t, double b, double a, double *P)
void PMAT_F81(double t, double u, double *BF, double *P)
void PMAT_F84(double t, double a, double b, double *BF, double *P)
void PMAT_HKY85(double t, double a, double b, double *BF, double *P)
void PMAT_T92(double t, double a, double b, double *BF, double *P)
void PMAT_TN93(double t, double a1, double a2, double b,
               double *BF, double *P)
void PMAT_GTR(double t, double a, double b,
              double c, double d, double e,
              double f, double *BF, double *P)
```

These eight functions have the same interface, and compute the 4×4 matrix of transition probabilities for various models of DNA evolution.

	On input	**On output**
`t`	the time interval	*unchanged*
`u`	the substition rate for JC69 or F81	*unchanged*
`a,b,c,d,e,f`	id. for the other models	*unchanged*
`*BF`	the base frequencies where applicable	*unchanged*
`*P`	an array of 16 doubles	the matrix of probabilities

3.3 Character Analysis

```
void pic(int *ntip, int *nnode, int *edge1, int *edge2,
         double *edge_len, double *phe, double *contr,
         double *var_contr, int *var, int *scaled)
```

Computes the phylogenetically independent contrasts of a continuous variable. The tree must be in pruningwise order.

	On input	On output
`*ntip`	the number of tips	*unchanged*
`*nnode`	the number of nodes	*unchanged*
`*edge1`	the first column of the `edge` matrix	*unchanged*
`*edge2`	the second column of the `edge` matrix	*unchanged*
`*edge_len`	the edge lengths	the rescaled edge lengths[a]
`*phe`	the `*ntip` observed phenotypes + `*nnode` doubles	*unchanged* for the first `*ntip` values + the estimated ancestral value for each node
`*contr`	`*nnode` doubles	the contrast for each node
`*var_contr`	an integer: 1 to compute the variances of the contrasts, 0 otherwise	*unchanged*
`*var`	`*nnode` doubles	the variances of the contrasts
`*scaled`	an integer: 1 to scale the contrasts, 0 otherwise	*unchanged*

[a] When `pic` is called with `.C`, the default `DUP = TRUE` is left, thus the data are duplicated before being sent to C and the original branch lengths in R are kept unchanged.

3.4 Numerical Utilities

```
void mat_expo(double *P, int *nr)
```

Computes the exponential of a square matrix (either symmetric or not).

	On input	On output
`*P`	the original matrix	its exponential
`*nr`	the number of rows of `*P`	*unchanged*

```
void mat_expo4x4(double *P)
```

Computes the exponential of a 4×4 matrix (either symmetric or not).

	On input	On output
`*P`	the original matrix made of 16 doubles	its exponential

```
double detFourByFour(double *x)
```

Computes directly the determinant of a 4×4 matrix.

Input	`*x`: an array of 16 doubles
Output	the determinant of `*x`

3.5 Miscellaneous

```
static int str2int(char *x, int n)
```

This function, currently not accessible outside the file 'tree_build.c', converts a character string of the form `"123"` into the integer 123. Note that the presence of characters other than `"0-9"` will lead to a spurious result without error, but leading zeros are accepted.

Input	`*x`: a character string
	`n`: the number of characters in `*x`
Output	the converted integer value

```
void decode_edge(const char *x, int a, int b,
                 int *node, double *w)
```

Reads inside a Newick string a series of characters of the form `"123:0.100"` so that `"123"` is converted into an integer, and `"0.100"` into a double.

	On input	**On output**
`*x`	a character string with the Newick tree	*unchanged*
`a,b`	the position of the first and last characters to be decoded inside `*x`	*unchanged*
`*node`	an integer	the value on the lhs of the colon (123 in the above example)
`*w`	a double	the value on the rhs of the colon (0.1 in the above example)

```
SEXP getListElement(SEXP list, char *str)
```

This function returns an element from a list `list` using the name stored in `*str`. It is borrowed from the *Writing R Extensions* manual and is not the same than then one found in 'library/stats/src/nls.c'.

```
int SameClade(SEXP clade1, SEXP clade2)
```

Compares two vectors of integers. It is assumed that both vectors are sorted in the same order; they may be of different lengths.

Input	`clade1, clade2`: an R vector of integers
Output	1 if `clade1` and `clade2` are identical, 0 otherwise

```
int whiteSpace(char c)
```

Returns 1 if `c` is a white space, a tabulation, or a newline, 0 otherwise.