

# Package: mmodely (via r-universe)

September 8, 2024

**Version** 0.2.5

**Date** 2023-05-05

**Title** Modeling Multivariate Origins Determinants - Evolutionary Lineages in Ecology

**Author** David M Schruth

**Maintainer** David M Schruth <dschruth@anthropoidea.org>

**Depends** R (>= 2.0.0),caper

**Imports** stats, caroline, ape

**Description** Perform multivariate modeling of evolved traits, with special attention to understanding the interplay of the multi-factorial determinants of their origins in complex ecological settings (Stephens, 2007 <[doi:10.1016/j.tree.2006.12.003](https://doi.org/10.1016/j.tree.2006.12.003)>). This software primarily concentrates on phylogenetic regression analysis, enabling implementation of tree transformation averaging and visualization functionality. Functions additionally support information theoretic approaches (Grueber, 2011 <[doi:10.1111/j.1420-9101.2010.02210.x](https://doi.org/10.1111/j.1420-9101.2010.02210.x)>; Garamszegi, 2011 <[doi:10.1007/s00265-010-1028-7](https://doi.org/10.1007/s00265-010-1028-7)>) such as model averaging and selection of phylogenetic models. Accessory functions are also implemented for coef standardization (Cade 2015), selection uncertainty, and variable importance (Burnham & Anderson 2000). There are other numerous functions for visualizing confounded variables, plotting phylogenetic trees, as well as reporting and exporting modeling results. Lastly, as challenges to ecology are inherently multifarious, and therefore often multi-dataset, this package features several functions to support the identification, interpolation, merging, and updating of missing data and outdated nomenclature.

**License** Apache License

**LazyLoad** yes

**NeedsCompilation** no

**Date/Publication** 2023-05-17 07:10:02 UTC

**Repository** <https://dranthropoid.r-universe.dev>

**RemoteUrl** <https://github.com/cran/mmodely>

**RemoteRef** HEAD

**RemoteSha** b434e85c5b6b0f2dedac7aa0cb87dfc4842755b6

## Contents

average.fit.models . . . . .	3
calc.q2n.ratio . . . . .	4
cept . . . . .	5
comp.data . . . . .	5
compare.data.gs.vs.tree.tips . . . . .	6
correct.AIC . . . . .	7
count.mod.vars . . . . .	7
ct.possible.models . . . . .	8
drop.na.data . . . . .	8
fit.ln.rprt . . . . .	9
get.mod.clmns . . . . .	10
get.mod.outcome . . . . .	11
get.mod.vars . . . . .	11
get.model.combos . . . . .	12
get.pgls.coefs . . . . .	12
get.phylo.stats . . . . .	13
gs.check . . . . .	14
gs.names.mismatch.check . . . . .	15
gs.rename . . . . .	16
interpolate . . . . .	17
missing.data . . . . .	17
missing.fill.in . . . . .	18
pgls.iter . . . . .	19
pgls.iter.stats . . . . .	20
pgls.print . . . . .	21
pgls.report . . . . .	22
pgls.wrap . . . . .	23
plot.confound.grid . . . . .	24
plot.pgls.iters . . . . .	25
plot.pgls.R2AIC . . . . .	26
plot.transformed.phylo . . . . .	27
plot.xy.ab.p . . . . .	28
select.best.models . . . . .	29
sparse.modsel . . . . .	30
trim.phylo . . . . .	31
weight.IC . . . . .	32

**Index**

**33**

---

average.fit.models      *Calculate a weighted average of pglm*

---

### Description

This function takes the output of `ppls.iter` and uses its list of objects model fits, optimizations (e.g. AICc) and performs a weighted average on the coefficients estimated in the former by weighting by the latter. The parameters can also optionally be converted to binary by specifying `binary=FALSE` or just running the alias wrapper function for assessing evidence of variable importance (Burnham & Anderson 2000).

### Usage

```
average.fit.models(vars, fits, optims, weight='AICw',
                  by=c('n', 'q', 'nXq', 'rwGsm')[1], round.digits=5, binary=FALSE, standardize=FALSE)
variable.importance(vars, fits, optims, weight='AICw',
                   by=c('n', 'q', 'nXq', 'rwGsm')[1], round.digits=5)
```

### Arguments

<code>vars</code>	variable names of model
<code>fits</code>	a list of PGLS model fits
<code>optims</code>	a list of PGLS optimization parameters (should include "AICw")
<code>weight</code>	a column name in the <code>optims</code> that specifies the weights to be used in the average
<code>by</code>	unique identifier used to group sub-datasets for reporting (defaults to <code>n</code> )
<code>round.digits</code>	the number of decimal places of the resultant mean to output
<code>binary</code>	converts all parameters to binary for presence or absence to calculate 'importance'
<code>standardize</code>	standardize the coefficient estimates by partial standard deviations, according to Cade (2015)

### Value

A vector of AICc difference weighted [AICw] averages of PGLS coefficients. Also returns model 'selection' errors or the square root of 'uncertainties' (Burnham & Anderson 2000)

### Examples

```
data.path <- system.file("extdata", "primate-example.data.csv", package="mmodely")
data <- read.csv(data.path, row.names=1)
pvs <- names(data[3:5])
data$gn_sp <- rownames(data)

tree.path <- system.file("extdata", "primate-springer.2012.tre", package="mmodely")
phyl <- ape::read.tree(tree.path)[[5]]
```

```

comp <- comp.data(phylo=phyl, df=data)

mods <- get.model.combos(predictor.vars=pvs, outcome.var='OC', min.q=2)

PGLSi <- pgls.iter(models=mods, phylo=phyl, df=data, k=1,l=1,d=1)

average.fit.models(vars=c('mass.Kg','group.size'), fits=PGLSi$fits, optim=PGLSi$optim)
variable.importance(vars=c('mass.Kg','group.size'), fits=PGLSi$fits, optim=PGLSi$optim)

```

---

calc.q2n.ratio

*Calculate the ratio of fit predictor variables to sample size*


---

### Description

The one in ten rule of thumb for model fitting suggest at least 10 fold as many data as parametes fit. This function allows for easily calculating that ratio on model selected PGLS fits.

### Usage

```
calc.q2n.ratio(coefs)
```

### Arguments

coefs                    a list of coefficients extracted from fit PGLS models

### Value

the ratio of q to n (on average for all extracted fit models)

### Examples

```

data.path <- system.file("extdata","primate-example.data.csv", package="mmodely")
data <- read.csv(data.path, row.names=1)
pvs <- names(data[3:5])
data$gn_sp <- rownames(data)

tree.path <- system.file("extdata","primate-springer.2012.tre", package="mmodely")
phyl <- ape::read.tree(tree.path)[[5]]

comp <- comp.data(phylo=phyl, df=data)

mods <- get.model.combos(predictor.vars=pvs, outcome.var='OC', min.q=2)

PGLSi <- pgls.iter(models=mods, phylo=phyl, df=data, k=1,l=1,d=1)

coefs.objs <- get.pgl.s.coefs(PGLSi$fits, est='Estimate')

calc.q2n.ratio(coefs.objs)

```

---

cept *Include all variables except ...*

---

### Description

This function takes a dataframe, list, or a named vector of variable (column) names to subset

### Usage

```
cept(x, except='gn_sp')
```

### Arguments

x                    a dataframe, list, or named vector  
 except              a vector of the names of the items in x to exclude

### Value

the subset of x without those 'except' items specified

### Examples

```
data.path <- system.file("extdata", "primate-example.data.csv", package="mmodely")
data <- read.csv(data.path, row.names=1)

df.except.gnsp <- cept(x=data, except='gn_sp')
```

---

comp.data *Comparative Data*

---

### Description

This is a shortcut function that wraps around "comparative.data" for use in the PGLS function.

### Usage

```
comp.data(phylo, df, gn_sp='gn_sp')
```

### Arguments

phylo                a tre file of the format phylo  
 df                   a data.frame with row names matching number and tip labels of 'tree'  
 gn\_sp                the column name (e.g. "gn\_sp") that indicates how to match df with tree

**Value**

a "comparative data" table for use in PGLS modeling

**Examples**

```
data.path <- system.file("extdata","primate-example.data.csv", package="mmodely")
data <- read.csv(data.path, row.names=1)
pvs <- names(data[3:5])
data$gn_sp <- rownames(data)

tree.path <- system.file("extdata","primate-springer.2012.tre", package="mmodely")
phyl <- ape::read.tree(tree.path)[[5]]

comp <- comp.data(phylo=phyl, df=data)
```

---

compare.data.gs.vs.tree.tips

*Find data being dropped by mismatches to the tree*

---

**Description**

This function simply lists the rows of the data that are not getting matched to tips of the tree.

**Usage**

```
compare.data.gs.vs.tree.tips(data, phylo, match.on=c('gn_sp', 'rownames')[1])
```

**Arguments**

data	a data frame with genus species information as row names and a column named "gn_sp"
phylo	a phylogenetic tree with labeled tip
match.on	use a character string specifying where the 'Genus_species' vector lies

**Value**

prints rows that are not matched ot the tree tips

**Examples**

```
data.path <- system.file("extdata","primate-example.data.csv", package="mmodely")
data <- read.csv(data.path, row.names=1)
data$gn_sp <- rownames(data)

tree.path <- system.file("extdata","primate-springer.2012.tre", package="mmodely")
phyl <- ape::read.tree(tree.path)[[5]]
```

```
compare.data.gs.vs.tree.tips(data, phyl, match.on='rownames')
```

---

correct.AIC	<i>Correct AIC</i>
-------------	--------------------

---

### Description

Calculate a corrected Akaike Information Criterion

### Usage

```
correct.AIC(AIC, K,n)
```

### Arguments

AIC	a vector of AIC values
K	number of parameters
n	number of data

### Value

corrected AIC values

### Examples

```
correct.AIC(AIC=100,K=10,n=100)
```

---

count.mod.vars	<i>Count the predictor variables in a model</i>
----------------	-------------------------------------------------

---

### Description

This function takes a model string and counts the number of predictor variables.

### Usage

```
count.mod.vars(model)
```

### Arguments

model	model specified as a string in the form "y ~ x1 + x2 ..."
-------	-----------------------------------------------------------

**Value**

an integer specifying the count of predictor variables

**Examples**

```
count <- count.mod.vars(model=formula('y ~ x1 + x2'))
if(count == 2) { print('sane'); }else{ print('insane')}
```

---

ct.possible.models	<i>Count all possible model combinations</i>
--------------------	----------------------------------------------

---

**Description**

Count all combinations of predictor variables in a multivariate regression model.

**Usage**

```
ct.possible.models(q)
```

**Arguments**

q                    number of predictor variables

**Value**

a count of the number of possible models

**Examples**

```
ct.possible.models(9)
```

---

drop.na.data	<i>Drop any rows with NA values</i>
--------------	-------------------------------------

---

**Description**

This function takes a dataframe as input and removes any rows that have NA as values.

**Usage**

```
drop.na.data(df, vars=names(df))
```



**Arguments**

df                    a dataframe  
 vars                sub set of variable (column) names to use in searching for missing values

**Value**

A subset of 'df' that only has non-missing values in the columns specified by 'vars'

**Examples**

```
path <- system.file("extdata","primate-example.data.csv", package="mmodely")
data <- read.csv(path, row.names=1)

df.nona <- drop.na.data(data, vars=names(df))
```

---

<code>fit.1ln.rprt</code>	<i>Report a model fit in a single line of text output</i>
---------------------------	-----------------------------------------------------------

---

**Description**

This function takes a fit multivariate regression model as input and converts the normal tabular output into a single line using repeated "+"or "-" symbols for significance

**Usage**

```
fit.1ln.rprt(fit, method=c('std.dev','p-value')[1], decimal.places=3,
             name.char.len=6, print.inline=TRUE, rtn.line=FALSE, R2AIC=TRUE, mn='')
```

**Arguments**

fit                    a fit model  
 method              how to calculate the number of pluses or minuses before each coefficient name  
 (default is standard deviations)  
 decimal.places      the number of decimal places to use in reporting p-values  
 name.char.len      the maximum length to use when truncating variable names  
 R2AIC                boolean for also returning/printing AIC and R<sup>2</sup> values  
 print.inline        should the outout string be printed to the terminal?  
 rtn.line            should the output string be returned as a characters string?  
 mn                    model number prefixed to printout if 'print.inline' is TRUE

**Value**

A character string of the form "++var1 +var5 var3 |-var2 -var4" indicating significance and direction of regression results

**Examples**

```

path <- system.file("extdata", "primate-example.data.csv", package="mmodely")
data <- read.csv(path, row.names=1)

model.fit <- lm('OC ~ mass.Kg + group.size + arboreal + leap.pct', data=data)

fit.ln.rprt(fit=model.fit, decimal.places=3, name.char.len=6, print.inline=TRUE, rtn.line=FALSE)

```

---

get.mod.clmns

*Get model columns*


---

**Description**

Get the variable names from a model string by splitting on "+" and '~' using both 'get.mod.outcome' and 'get.mod.vars'. The results are passed to the comp.data function for eventual use in PGLS modeling. 'gn\_sp' is included as it is typically required to link tree tips to rows of the comparative data.

**Usage**

```
get.mod.clmns(model, gs.clmn='gn_sp')
```

**Arguments**

model	a model string of the form "y ~ x1 + x2 ..."
gs.clmn	the column header for the vector of "Genus_species" names, to link a tree tips to rows

**Value**

a vector of characters enumerating the columns to retain in PGLS modeling (input to df param in the 'comp.data' function)

**Examples**

```
model.columns <- get.mod.clmns(model=formula('y ~ x1 + x2'))
```

---

get.mod.outcome	<i>Get the outcome variable from a model string</i>
-----------------	-----------------------------------------------------

---

**Description**

Get the outcome variable from the front of a model formula string. Used as part of 'get.mod.clmns' to be passed to 'comp.data'

**Usage**

```
get.mod.outcome(model)
```

**Arguments**

model            a character string of a formula of the form 'y ~ x1 + x2 ...'

**Value**

a character string specifying the outcome variable

**Examples**

```
model.columns <- get.mod.clmns(model=formula('y ~ x1 + x2'))
```

---

get.mod.vars	<i>Get model variable names</i>
--------------	---------------------------------

---

**Description**

Split the predictor string of a model formula into it's constituent character strings.

**Usage**

```
get.mod.vars(model)
```

**Arguments**

model            a character string of a formula of the form 'y ~ x1 + x2'

**Value**

a vector of character strings of variable names (e.g. corresponding to column names for comp.data input)

**Examples**

```
model.variables <- get.mod.vars(model='y ~ x1 + x2')
```

---

`get.model.combos`      *All combinations of predictor variables*

---

### Description

Enumerate all combinations of predictor variables in a multivariate regression model.

### Usage

```
get.model.combos(outcome.var, predictor.vars, min.q=1)
```

### Arguments

`predictor.vars` predictor variables names (a vector of character strings)  
`outcome.var` outcome variable name (character string)  
`min.q` minimum number of predictor variables to include in the mode (default is 2)

### Value

a vector of models as character strings of the form "y ~ x1 + x2 ..."

### Examples

```
path <- system.file("extdata", "primate-example.data.csv", package="mmodely")
data <- read.csv(path, row.names=1)

get.model.combos(outcome.var='OC', predictor.vars=names(data), min.q=2)
```

---

`get.pgls.coefs`      *Get coefficients from a list of PGLS model-fits (from each selected subset)*

---

### Description

Post PGLS model selection, the list of all possible PGLS model fits can be subset and passed to this function, which harvests out the coefficients or t-values for each model into bins for the coefficients

### Usage

```
get.pgls.coefs(pgls.fits, est=c("t value", "Estimate", "Pr(>|t|)")[1])
```

### Arguments

`pgls.fits` a list of PGLS models output from 'pgls' or 'pgls.report'  
`est` a character string indicating if Estimate or t value should be used as data points in the plot, default is 'Estimate'

**Value**

A list of PGLS coefficients (lists of estimates and t-values) organized by coefficient-named bins

**Examples**

```
data.path <- system.file("extdata","primate-example.data.csv", package="mmodely")
data <- read.csv(data.path, row.names=1)
pvs <- names(data[3:5])
data$gn_sp <- rownames(data)

tree.path <- system.file("extdata","primate-springer.2012.tre", package="mmodely")
phyl <- ape::read.tree(tree.path)[[5]]

comp <- comp.data(phylo=phyl, df=data)

mods <- get.model.combos(predictor.vars=pvs, outcome.var='OC', min.q=2)

PGLSi <- pgls.iter(models=mods, phylo=phyl, df=data, k=1,l=1,d=1)

coefs.objs <- get.pgl.s.coefs(PGLSi$fits, est='Estimate')
```

---

<code>get.phylo.stats</code>	<i>Get tree statistics for a trait</i>
------------------------------	----------------------------------------

---

**Description**

This function uses Pagel's lambda, Blombergs k, and Ancestral Character Estimation [ACE] to calculate statistics on a tree given a specified trait.

**Usage**

```
get.phylo.stats(phylo, data, trait.clmn, gs.clmn='gn_sp',
               ace.method='REML', ace.scaled=TRUE, ace.kappa=1)
```

**Arguments**

<code>phylo</code>	PARAMDESCRIPTION
<code>data</code>	PARAMDESCRIPTION
<code>trait.clmn</code>	PARAMDESCRIPTION
<code>gs.clmn</code>	PARAMDESCRIPTION
<code>ace.method</code>	PARAMDESCRIPTION
<code>ace.scaled</code>	PARAMDESCRIPTION
<code>ace.kappa</code>	PARAMDESCRIPTION

**Value**

statistics on a particular trait within a tree (Pagel's lambda, Blomberg's k, and the most ancestral ACE estimate)

**Examples**

```
data.path <- system.file("extdata", "primate-example.data.csv", package="mmodely")
data <- read.csv(data.path, row.names=1)

data$gn_sp <- rownames(data)

tree.path <- system.file("extdata", "primate-springer.2012.tre", package="mmodely")
phyl <- ape::read.tree(tree.path)[[5]]

get.phylo.stats(phylo=phyl, data=data, trait.clmn='OC',
               gs.clmn='gn_sp', ace.method='REML', ace.scaled=TRUE, ace.kappa=1)
```

---

 gs.check

---

*Check "Genus species" name formatting*


---

**Description**

This convenience function checks to make sure that all of the elements the provided character vector adhere to the "Genus species" naming convention format. Default delimiters between genus and species names in the string are " ", "\_", or "."

**Usage**

```
gs.check(genus.species, sep='[_\\. ]')
```

**Arguments**

genus.species a vector of character strings specifying the combination of Genus [and] species  
 sep a regular expression between genus and species

**Value**

None

**Examples**

```
path <- system.file("extdata", "primate-example.data.csv", package="mmodely")
gs.tab <- read.csv(path, row.names=1)
gs.tab$gn_sp <- rownames(gs.tab)

gs.check(genus.species=gs.tab$gn_sp, sep='[_\\. ]')
```

---

`gs.names.mismatch.check`*Check "Genus species" name formatting*

---

### Description

This convenience function checks to make sure that all of the elements the provided character vector adhere to the "Genus species" naming convention format. Default delimiters between genus and species names in the string are " ", "\_", or "."

### Usage

```
gs.names.mismatch.check(df, alias.table.path, gs.clmn='gn_sp')
```

### Arguments

`df` a data frame with genus species information as row names and optionally in a column named "gn\_sp"

`alias.table.path` a file system path (e.g. 'inst/extdata/primate.taxa.aliases.tab') to a lookup table with 'old.name' and 'new.name' as columns

`gs.clmn` the name of the column containing the 'Genus\_species' vector

### Value

None

### Examples

```
path <- system.file("extdata", "primate-example.data.csv", package="mmodely")
gs.tab <- read.csv(path, row.names=1)
gs.tab$gn_sp <- rownames(gs.tab)

path.look <- system.file("extdata", "primate.taxa.aliases.tab", package="mmodely")

gs.names.mismatch.check(gs.tab, alias.table.path=path.look, gs.clmn='gn_sp')
```

---

gs.rename	<i>Rename the Genus species information in a data frame</i>
-----------	-------------------------------------------------------------

---

### Description

This function takes a data frame (with a genus species column) and proceeds to use an external look-up table to update the names if they've been changed

### Usage

```
gs.rename(df, alias.table.path, retro=FALSE, update.gn_sp=FALSE)
```

### Arguments

df	a data frame with genus species information as row names and optionally in a column named "gn_sp"
alias.table.path	a file system path (e.g. 'inst/extdata/primate.taxa.aliases.tab') to a lookup table with 'old.name' and 'new.name' as columns
retro	a boolean (T/F) parameter specifying if the renaming should go from new to old instead of the default of old to new
update.gn_sp	a boolean parameter specifying if the 'gn_sp' column should also be updated with 'new.name's

### Value

the original data frame with (potentially) updated row names and updated gn\_sp column values

### Examples

```
path.data <- system.file("extdata","primate-example.data.csv", package="mmodely")
data <- read.csv(path.data, row.names=1)

path.look <- system.file("extdata","primate.taxa.aliases.tab", package="mmodely")

data.renamed <- gs.rename(df=data, alias.table.path=path.look, retro=FALSE, update.gn_sp=FALSE)
```



---

interpolate	<i>Interpolate missing data in a data frame</i>
-------------	-------------------------------------------------

---

**Description**

This function finds NA values and interpolates using averaging values of nearby genus and species

**Usage**

```
interpolate(df, taxa=c('genus','family'), c1mns=1:length(df))
```

**Arguments**

df	a data frame
taxa	a vector of taxonomic ranks (corresponding to columns) to assist in guiding the interpolating
c1mns	the names of the columns to interpolate over

**Value**

a modified data frame without missing values in the columns specified

**Examples**

```
path <- system.file("extdata", "primate-example.data.csv", package="mmodely")
gs.tab <- read.csv(path, row.names=1)

c1mns <- match(c('mass.Kg', 'DPL.km'), names(gs.tab))
df.2 <- interpolate(df=gs.tab, taxa='genus', c1mns=c1mns)
```

---

missing.data	<i>Report missing values in a dataframe</i>
--------------	---------------------------------------------

---

**Description**

This function reports column and rowwise missing data. It can also list the rownames for missing columns or the column names for missing rows.

**Usage**

```
missing.data(x, cols=NULL, rows=NULL)
```

**Arguments**

x	a dataframe
cols	print the specific rows corresponding to missing values in this column
rows	print the specific cols corresponding to missing values in this rowname

**Value**

a report on column versus row wise missing data

**Examples**

```
path <- system.file("extdata", "primate-example.data.csv", package="mmodely")
data <- read.csv(path, row.names=1)

missing.data(data)
```

---

missing.fill.in

*Fill in missing values in a dataframe with a secondary source*

---

**Description**

This function uses the (non-missing) values from one column to fill in the missing values of another

**Usage**

```
missing.fill.in(x, var.from, var.to)
```

**Arguments**

x	a dataframe or matrix
var.from	secondary variable (of the same type and units) providing values to 'var.to'
var.to	primary variable with missing values to fill in by 'var.from'

**Value**

a modified dataframe with fewer missing values in the 'var.to' column

**Examples**

```
df <- data.frame(a=c(1,2,NA),b=c(1,NA,3),c=c(1,2,6))
missing.fill.in(df, 'c', 'a')
```

---

pgls.iter *Iterate through PGLS estimations*

---

### Description

This function takes phylogenetic tree and a list of (all possible) combinations of variables as a vector of model strings and estimates PGLS fits based on the bounds or tree parameters provided separately.

### Usage

```
pgls.iter(models, phylo, df, gs.clmn='gn_sp',
          b=list(lambda=c(.2,1),kappa=c(.2,2.8),delta=c(.2,2.8)),l='ML', k='ML',d='ML')
```

### Arguments

models	a vector of all possible model formulas (as character strings)
phylo	a phylogenetic tree
df	the name of the column used to specify 'Genus_species'
gs.clmn	the name of the column containing the 'Genus_species' vector
b	a list of vectors of upper and lower bounds for kappa, lambda, and delta
k	the fixed or 'ML' value for kappa
l	the fixed or 'ML' value for lambda
d	the fixed or 'ML' value for delta

### Value

a list of fit PGLS regression models plus 'optim' and 'param' support tables

### Examples

```
data.path <- system.file("extdata","primate-example.data.csv", package="mmodely")
data <- read.csv(data.path, row.names=1)
pvs <- names(data[3:5])
data$gn_sp <- rownames(data)

tree.path <- system.file("extdata","primate-springer.2012.tre", package="mmodely")
phyl <- ape::read.tree(tree.path)[[5]]

comp <- comp.data(phylo=phyl, df=data)

mods <- get.model.combos(predictor.vars=pvs, outcome.var='OC', min.q=2)

PGLSi <- pgls.iter(models=mods, phylo=phyl, df=data, k=1,l=1,d=1)
```

---

pgls.iter.stats      *Statistics from PGLS runs*

---

### Description

Print (and plot) statistics from a list of PGLSs fitted models and tables of associated parameters.

### Usage

```
pgls.iter.stats(PGLSi, verbose=TRUE, plots=FALSE)
```

### Arguments

PGLSi	a list of PGLS iter objects, each of which is list including: fitted PGLS model, a optim table, and a tree-transformation parameter table
verbose	the model formula (as a character string)
plots	the fixed or 'ML' value for kappa

### Value

A summary statistics on each of the objects in the PGLS list of lists

### Examples

```
data.path <- system.file("extdata", "primate-example.data.csv", package="mmodely")
data <- read.csv(data.path, row.names=1)
pvs <- names(data[3:5])
data$gn_sp <- rownames(data)

tree.path <- system.file("extdata", "primate-springer.2012.tre", package="mmodely")
phyl <- ape::read.tree(tree.path)[[5]]

comp <- comp.data(phylo=phyl, df=data)

mods <- get.model.combos(predictor.vars=pvs, outcome.var='OC', min.q=2)

PGLSi <- pgls.iter(models=mods, phylo=phyl, df=data, k=1,l=1,d=1)

pgls.iter.stats(PGLSi, verbose=TRUE, plots=FALSE)
```

---

pgls.print                      *Print the results of a PGLS model fit*

---

### Description

Print the results of a PGLS model fit

### Usage

```
pgls.print(pgls, all.vars=names(pgls$data$data)[-1],
           model.no=NA, mtx.out=NA, write=TRUE, print=FALSE)
```

### Arguments

pgls	a fit PGLS model
all.vars	the names of all the variables to be reported
model.no	the model number (can be the order that models were run)
mtx.out	should a matrix of the tabular summary results be returned
write	should the matrix of summary results be written to disk?
print	should the matrix of summary results be printed to screen?

### Value

A matrix of summary results of a fit PGLS model

### Examples

```
data.path <- system.file("extdata","primate-example.data.csv", package="mmodely")
data <- read.csv(data.path, row.names=1)
data$gn_sp <- rownames(data)

tree.path <- system.file("extdata","primate-springer.2012.tre", package="mmodely")

#5. RAxML phylogram based on the 61199 bp concatenation of 69 nuclear and ten mitochondrial genes.
phyl <- ape::read.tree(tree.path)[[5]]

phyl <- trim.phylo(phylo=phyl, gs.vect=data$gn_sp)

comp <- comp.data(phylo=phyl, df=data)

a.PGLS <- caper::pgls(formula('OC~mass.Kg + DPL.km'), data=comp)

pgls.print(a.PGLS, all.vars=names(a.PGLS$data$data)[-1],
           model.no=NA, mtx.out='', write=FALSE, print=FALSE)
```

---

pgls.report

*Report PGLS results as a table*

---

## Description

Output a spreadsheet ready tabular summary of a fit PGLS model

## Usage

```
pgls.report(cd, f=formula('y~x'), l=1,k=1,d=1,
            bounds=list(lambda=c(.2,1),kappa=c(.2,2.7),delta=c(.2,2.7)),
            anova=FALSE, mod.no='NA', out='pgls.output-temp',QC.plot=FALSE)
```

## Arguments

cd	a comparative data object, here created by 'comp.data'
f	the model formula (as a character string)
k	the fixed or 'ML' value for kappa
l	the fixed or 'ML' value for lambda
d	the fixed or 'ML' value for delta
bounds	a list of vectors of upper and lower bounds for kappa, lambda, and delta
anova	should an anova be run on the fit model and output to the terminal?
mod.no	the model number (can be the order that models were run)
out	the base filename to be printed out
QC.plot	should a quality control plot be output to screen?

## Value

A summary results of a fit PGLS model with ANOVA and tabular spreadsheet ready csv filesystem output.

## Examples

```
data.path <- system.file("extdata","primate-example.data.csv", package="mmodely")
data <- read.csv(data.path, row.names=1)
data$gn_sp <- rownames(data)

tree.path <- system.file("extdata","primate-springer.2012.tre", package="mmodely")
#5. RAXML phylogram based on the 61199 bp concatenation of 69 nuclear and ten mitochondrial genes.
phyl <- ape::read.tree(tree.path)[[5]]

phyl <- trim.phylo(phylo=phyl, gs.vect=data$gn_sp)

comp <- comp.data(phylo=phyl, df=data)
```

```
pgls.report(comp, f=formula('OC~mass.Kg + DPL.km'), l=1,k=1,d=1,
            anova=FALSE, mod.no='555', out='', QC.plot=TRUE)
```

---

pgls.wrap

*A Wrapper for PGLS model*


---

## Description

Print the results of an unfit PGLS model

## Usage

```
pgls.wrap(cd, f, b, l, k, d, all.vars=names(cd$data)[-1],
          model.no=NA, mtx.out=NA, write=TRUE, print=FALSE)
```

## Arguments

cd	a 'comparative data' object, here created by 'comp.data(phylo, df, gs.clmn)'
f	the model formula (as a character string)
b	a list of vectors of upper and lower bounds for kappa, lambda, and delta
l	the fixed or 'ML' value for lambda
k	the fixed or 'ML' value for kappa
d	the fixed or 'ML' value for delta
all.vars	the names of all the variables to be reported
model.no	the model number (can be the order that models were run)
mtx.out	should a matrix of the tabular summary results be returned
write	should the matrix of summary results be written to disk?
print	should the matrix of summary results be printed to screen?

## Value

A matrix of summary results of a fit PGLS model

## Examples

```
data.path <- system.file("extdata", "primate-example.data.csv", package="mmodely")
data <- read.csv(data.path, row.names=1)
data$gn_sp <- rownames(data)

tree.path <- system.file("extdata", "primate-springer.2012.tre", package="mmodely")
#5. RAxML phylogram based on the 61199 bp concatenation of 69 nuclear and ten mitochondrial genes.
phyl <- ape::read.tree(tree.path)[[5]]

phyl <- trim.phylo(phylo=phyl, gs.vect=data$gn_sp)
```

```

comp <- comp.data(phylo=phyl, df=data)

model <- 'OC ~ mass.Kg + group.size'

pgls.wrap(cd=comp, f=model, b=list(kappa=c(.3, 3), lambda=c(.3, 3), delta=c(.3, 3)),
          l=1, k=1, d=1, all.vars=names(cd.obj$data)[-1])

```

---

`plot.confound.grid`      *Plot a grid of x y plots split by a confounder z*

---

### Description

Plot a grid of x y plots showing how a third confounding variable 'z' changes the slope

### Usage

```

## S3 method for class 'confound.grid'
plot(x, Y='y', X='x', confounder='z', breaks=3, ...)

```

### Arguments

<code>x</code>	a data frame
<code>Y</code>	the name of the column with the dependent/outcome variable
<code>X</code>	the name of the column with the predictor variable
<code>confounder</code>	the name of the column with confounding variable
<code>breaks</code>	number or vector of breaks to split the plots horizontally (across x)
<code>...</code>	other arguments passed to 'plot'

### Value

a confound grid plot

### Examples

```

path <- system.file("extdata", "primate-example.data.csv", package="mmodely")
data <- read.csv(path, row.names=1)
data$col <- c('yellow', 'red')[data$nocturnal+1]

plot.confound.grid(x=data, Y='OC', X='leap.pct', confounder='mass.Kg')

```



---

plot.pgls.iters                    *Plot the PGLS iterations*

---

### Description

A plot of AIC (and AICc) vs  $R^2$  (and adjusted  $R^2$ ) for all of the PGLS iterations

### Usage

```
## S3 method for class 'pgls.iters'
plot(x,
      bests=bestBy(x$optim, by=c('n', 'q', 'qXn', 'rwGsm')[1], best=c('AICc', 'R2.adj')[1],
                    inverse=FALSE), ...)
```

### Arguments

x	a PGLSi[iteration] object (a list of pgls model fits as well as optimization and tree parameter tables)
bests	a table of the 'best' models to highlight in the plot based on some optimization criterion (e.g. R2)
...	other parameters passed to 'plot'

### Value

a plot of all of PGLS iterations

### Examples

```
data.path <- system.file("extdata", "primate-example.data.csv", package="mmodely")
data <- read.csv(data.path, row.names=1)
pvs <- names(data[3:5])
data$gn_sp <- rownames(data)

tree.path <- system.file("extdata", "primate-springer.2012.tre", package="mmodely")
phyl <- ape::read.tree(tree.path)[[5]]

mods <- get.model.combos(predictor.vars=pvs, outcome.var='OC', min.q=2)

PGLSi <- pgls.iter(models=mods, phylo=phyl, df=data, k=1,l=1,d=1)

# sprinkle in some missing data so as to make model selection more interesting
for(pv in pvs){ data[sample(x=1:nrow(data),size=2),pv] <- NA}

PGLSi <- pgls.iter(models=mods, phylo=phyl, df=data, k=1,l=1,d=1)

# find the lowest AIC within each q by n sized sub-dataset
plot.pgls.iters(x=PGLSi)
```

---

plot.pgls.R2AIC      *Plot (R2 vs AIC) results of a collection of fit PGLS models*

---

### Description

Plots a single panel of  $R^2$  versus AIC, using versions of your choosing.

### Usage

```
## S3 method for class 'pgls.R2AIC'
plot(x,
     bests=bestBy(x, by=c('n', 'q', 'qXn', 'rwGsm')[4], best=c('AICc', 'R2.adj')[1],
                   inverse=c(FALSE, TRUE)[1]), bcl=rgb(1,1,1,maxColorValue=3,alpha=1), nx=2,
     model.as.title='', ...)
```

### Arguments

x	a PGLSi[teration]\$optim [optimization] table
bests	a list of the best PGLS models grouped by variable count and sorted by some metric (e.g. adjusted R2)
bcl	background color of plot point
nx	point size expansion factor to multiply against sample size ratio (this model to max of all models)
model.as.title	uses model.ln.report to create a short character string of the "best" model results as a title
...	other parameters passed to 'plot'

### Value

a plot of  $R^2$  versus AIC of many PGLS models

### Examples

```
data.path <- system.file("extdata", "primate-example.data.csv", package="mmodely")
data <- read.csv(data.path, row.names=1)
pvs <- names(data[3:6])
data$gn_sp <- rownames(data)

tree.path <- system.file("extdata", "primate-springer.2012.tre", package="mmodely")
phyl <- ape::read.tree(tree.path)[[5]]

mods <- get.model.combos(predictor.vars=pvs, outcome.var='OC', min.q=2)

# sprinkle in some missing data so as to make model selection more interesting
for(pv in pvs){ data[sample(x=1:nrow(data), size=2), pv] <- NA}

PGLSi <- pgls.iter(models=mods, phylo=phyl, df=data, k=1, l=1, d=1)
```

```
plot.pgls.R2AIC(PGLSi$optim) # find the lowest AIC within each q by n sized sub-dataset
```

---

```
plot.transformed.phylo
```

*Plot a transformed phylogenetic tree*

---

### Description

PGLS regression will use maximum likelihood to estimate tree parameters while also estimating regression parameters. Here we provide a utility function to visualize what this new tree would look like in two dimensions.

### Usage

```
## S3 method for class 'transformed.phylo'  
plot(x, delta=1, kappa=1, ...)
```

### Arguments

x	a phylogenetic tree
delta	an integer between 0 and 3
kappa	an integer between 0 and 3
...	other parameters passed to 'plot'

### Value

a plot of a transformed phylogenetic tree

### Examples

```
tree.path <- system.file("extdata", "primate-springer.2012.tre", package="mmodely")  
phyl <- ape::read.tree(tree.path)[[5]]  
  
plot.transformed.phylo(x=phyl, delta=2.3, kappa=2.1)
```

---

`plot.xy.ab.p`*An x/y scatterplot with a linear regression line and p-value*

---

### Description

This function performs a simple scatter plot but also superimposes a linear regression trend (abline) and optionally also the p-value of this line

### Usage

```
## S3 method for class 'xy.ab.p'  
plot(x, x.var, y.var,  
      fit.line=TRUE, p.value=TRUE, slope=TRUE, p.col='red', plot.labels=TRUE, verbose=TRUE, ...)
```

### Arguments

<code>x</code>	a data frame
<code>x.var</code>	the name of the x variable in df
<code>y.var</code>	the name of the y variable in df
<code>fit.line</code>	should a fit (ab) line be drawn?
<code>p.value</code>	should the p-value be printed on the plot?
<code>slope</code>	should the slope be printed on the plot?
<code>p.col</code>	should the plot be labeled?
<code>plot.labels</code>	should all of this model fit information be printed out?
<code>verbose</code>	should all other information be printed out too?
<code>...</code>	other parameters passed to 'plot'

### Value

An x/y scatterplot with regression line

### Examples

```
path <- system.file("extdata", "primate-example.data.csv", package="mmodely")  
data <- read.csv(path, row.names=1)  
  
plot.xy.ab.p(x=data, x.var='OC', y.var='group.size',  
             fit.line=TRUE, p.value=TRUE, slope=TRUE, p.col='red', plot.labels=TRUE, verbose=TRUE)
```

---

select.best.models      *Get the best model from list of PGLS model fits*

---

### Description

Get the outcome variable from the front of a model formula string. Used as part of 'get.mod.clmns' to be passed to 'comp.data'

### Usage

```
select.best.models(PGLSi, using=c('AICc', 'R2.adj', 'AIC', 'R2')[1],
                  by=c('n', 'q', 'nXq', 'rwGsm')[1])
```

### Arguments

PGLSi	a list of PGLS iter objects, each of which is list including: fitted PGLS model, a optim table, and a tree-transformation parameter table
using	performance metric to use in searching for the best model
by	unique identifier used to group sub-datasets for reporting (defaults to n)

### Value

a line corresponding to the "best" models from the PGLSi "optim" table

### Examples

```
data.path <- system.file("extdata", "primate-example.data.csv", package="mmodely")
data <- read.csv(data.path, row.names=1)
pvs <- names(data[3:5])
data$gn_sp <- rownames(data)

tree.path <- system.file("extdata", "primate-springer.2012.tre", package="mmodely")
phyl <- ape::read.tree(tree.path)[[5]]

comp <- comp.data(phylo=phyl, df=data)

mods <- get.model.combos(predictor.vars=pvs, outcome.var='OC', min.q=2)

PGLSi <- pglS.iter(models=mods, phylo=phyl, df=data, k=1, l=1, d=1)

a.PGLS <- select.best.models(PGLSi, by=c('R2.adj', 'AICc')[1])
```

---

sparge.modsel	<i>Coefficients distribution [sparge] plot of models selected from each subset</i>
---------------	------------------------------------------------------------------------------------

---

### Description

Plot the raw distribution of points corresponding to the coefficients harvested from the best model of each subset of the dataset.

### Usage

```
sparge.modsel(PC, jit.f=1, R2x=3, nx=2, n.max=max(unlist(PC$n)), zeroline=TRUE,
              add=FALSE, pd=0, pvs=names(PC$coefs), pvlabs=NULL,
              xlim=range(unlist(PC$coefs)),
              MA = NULL, ap=8, ac = 1, ax = nx, ...)
```

### Arguments

PC	a list of vectors of pooled coefficients (or scores) harvested from the 'best' selected modeling runs (out put from 'get.ppls.coefs')
jit.f	factor for random jittering (see 'jitter()')
R2x	the line width expansion factor according to R <sup>2</sup> value
nx	the point size expansion factor according to sample size of model
n.max	the maximum sample size used in all models
zeroline	should we add an abline at x=0?
add	should we add to the existing plot?
pd	'position dodge' moves all y axis plotting positions up or down by this provided value (useful for adding multiple distributions for the same param)
pvs	the predictor variable vector for ordering the y-axis labels
pvlabs	the predictor variable labels for labeling the plot (defaults to pvs)
xlim	x axis plot limits
MA	matrix of model averages (defaults to NULL)
ap	coded numeric point character symbol used for model averaged parameter position
ac	color symbol used for model averaged parameters plot character
ax	expansion factor to expand model average parameter plot character (defaults to nx)
...	other parameters passed on to plot

### Value

a 'sparge' [sprinkle/smear] plot of coefficient distributions

**See Also**

See also 'boxplot' and 'stripchart' in package 'graphics' as well as 'violin', 'bean', 'ridgelines', and 'raincloud' plots.

**Examples**

```
data.path <- system.file("extdata","primate-example.data.csv", package="mmodely")
data <- read.csv(data.path, row.names=1)
pvs <- names(data[3:5])
data$gn_sp <- rownames(data)

tree.path <- system.file("extdata","primate-springer.2012.tre", package="mmodely")
phyl <- ape::read.tree(tree.path)[[5]]

mods <- get.model.combos(predictor.vars=pvs, outcome.var='OC', min.q=2)

PGLSi <- pglis.iter(models=mods, phylo=phyl, df=data, k=1,l=1,d=1)

coefs.objs <- get.pglis.coefs(PGLSi$fits, est='Estimate')

sparge.modsel(coefs.objs)
```

---

trim.phylo

*Trim a phylogenetic tree using Genus species names*

---

**Description**

Read in a vector of genus species names and a tree and drop the tips in the tree that match the vector of names.

**Usage**

```
trim.phylo(phylo, gs.vect)
```

**Arguments**

phylo	a phylogenetic tree
gs.vect	a vector of character strings in the 'Genus_species' format

**Value**

a plot of a transformed phylogenetic tree

**Examples**

```

data.path <- system.file("extdata","primate-example.data.csv", package="mmodely")
data <- read.csv(data.path, row.names=1)
data$gn_sp <- rownames(data)

tree.path <- system.file("extdata","primate-springer.2012.tre", package="mmodely")
phylo <- read.tree(tree.path)[[5]]

trim.phylo(phylo, gs.vect=data$gn_sp)

```

weight.IC

*Get IC weights***Description**

An implementation of IC weighting that first calculates the difference in IC values by subtracting all values from the lowest IC value. Second, the changes are exponentiated divided by a sum of the same and exponentiated yet again.

**Usage**

```
weight.IC(IC)
```

**Arguments**

IC                    a vector of IC values

**Value**

a vector of IC based weights

**Examples**

```

data.path <- system.file("extdata","primate-example.data.csv", package="mmodely")
data <- read.csv(data.path, row.names=1)
pvs <- names(data[3:5])
data$gn_sp <- rownames(data)

tree.path <- system.file("extdata","primate-springer.2012.tre", package="mmodely")
phyl <- ape::read.tree(tree.path)[[5]]

comp <- comp.data(phylo=phyl, df=data)

mods <- get.model.combos(predictor.vars=pvs, outcome.var='OC', min.q=2)

PGLSi <- pglis.iter(models=mods, phylo=phyl, df=data, k=1,l=1,d=1)

AICc.w <- weight.IC(IC=PGLSi$optim$AICc)

```



# Index

average.fit.models, 3

calc.q2n.ratio, 4

cept, 5

comp.data, 5

compare.data.gs.vs.tree.tips, 6

correct.AIC, 7

count.mod.vars, 7

ct.possible.models, 8

drop.na.data, 8

fit.1ln.rprt, 9

get.mod.clmns, 10

get.mod.outcome, 11

get.mod.vars, 11

get.model.combos, 12

get.ppls.coefs, 12

get.phylo.stats, 13

gs.check, 14

gs.names.mismatch.check, 15

gs.rename, 16

interpolate, 17

missing.data, 17

missing.fill.in, 18

ppls.iter, 19

ppls.iter.stats, 20

ppls.print, 21

ppls.report, 22

ppls.wrap, 23

plot.confound.grid, 24

plot.ppls.iters, 25

plot.ppls.R2AIC, 26

plot.transformed.phylo, 27

plot.xy.ab.p, 28

select.best.models, 29

sparge.modsel, 30

trim.phylo, 31

variable.importance  
(average.fit.models), 3

weight.IC, 32