Package 'Pstat'

May 7, 2026

Type Package

Title Assessing Pst Statistics

Version 1.2

Date 2017-11-04

Author Blondeau Da Silva Stephane [aut, cre] - Da Silva Anne [aut].

Maintainer Blondeau Da Silva Stephane <blondeaudasilva@xlim.fr>

Description Calculating Pst values to assess differentiation among populations from a set of quantitative traits is the primary purpose of such a package. The bootstrap method provides confidence intervals and distribution histograms of Pst. Variations of Pst in function of the parameter c/h^2 are studied as well. Finally, the package proposes different transformations especially to eliminate any variation resulting from allometric growth (calculation of residuals from linear regressions, Reist standardizations or Aitchison transformation).

License GPL-2

NeedsCompilation no

Repository CRAN

Date/Publication 2017-11-05 07:28:16 UTC

Contents

Pstat-package 2
AitTrans 3
BootPst 4
Pst 10
ReistTrans 15
Res 17
test 19
TracePst 20

Index **27**

Pstat-package *Assessing Pst Statistics*

Description

This package aims at calculating Pst values to assess differentiation among populations from a set of quantitative traits (with the function 'Pst'). The bootstrap method provides confidence intervals and distribution histograms of Pst (with the function 'BootPst'). Variations of Pst in function of the parameter c/h^2 are studied (with the function 'TracePst') as well. Moreover the package allows users to transform theirs datas in three different ways in particular to eliminate any variation resulting from allometric growth (calculation of residuals from linear regressions -with the function 'Res'-, Reist standardizations -with the function 'ReistTrans'- or Aitchison transformation -with the function 'AitTrans'-).

Details

The DESCRIPTION file: This package aims at calculating Pst values to assess differentiation among populations from a set of quantitative traits (with the function 'Pst'). The bootstrap method provides confidence intervals and distribution histograms of Pst (with the function 'BootPst'). Variations of Pst in function of the parameter c/h^2 are studied (with the function 'TracePst') as well. Moreover the package allows users to transform theirs datas in three different ways in particular to eliminate any variation resulting from allometric growth (calculation of residuals from linear regressions -with the function 'Res'-, Reist standardizations -with the function 'ReistTrans'- or Aitchison transformation -with the function 'AitTrans'-).

Author(s)

Blondeau Da Silva Stephane [aut, cre] - Da Silva Anne [aut].

Maintainer: Blondeau Da Silva Stephane <blondeaudasilva@xlim.fr>

References

Spitze K., 1993. Population structure in Daphnia obtusa: Quantitative genetic and allozymic variation. Genetics 135: 367-374.

Merila J., Crnokrak P., 2001. Comparison of genetic differentiation at marker loci and quantitative traits. Journal Evolution Biology 14: 892-903.

Leinonen T., O'Hara R.B., Cano J.M., Merila J., 2008. Comparative studies of quantitative trait and neutral marker divergence: a meta-analysis. Journal Evolution Biology 21: 1-17.

Brommer J.E., 2011. Whither Pst? The approximation of Qst by Pst in evolutionary and conservation biology. Journal Evolution Biology 24: 1160-1168.

Examples

```
# data(test)
# test1=AitTrans(test)
# Pst(test1,csh=0.2,ci=1)
```

```
# test2=ReistTrans(test,reg="Body_length")
# BootPst(test2,va="QM4",opt="hist",Ri=c(3,7:17),Pw=c("C","D"),bars=50)
# TracePst(test2,va=c(7,9:12),Fst=0.3,Ri=c(22,27,195),Rp=c("A","C","E"))
```

`AitTrans` *Aitchison transformation*

Description

'AitTrans' calculates log-ratio transformed measurements to account for individual size-effects.

Usage

```
AitTrans(data, Rp = 0, Ri = 0)
```

Arguments

`data`	a dataframe with as many rows as individuals. The first column contains the name of the population to which the individual belongs, the others contain quantitative variables.
`Rp`	a vector containing the names of the populations to be deleted.
`Ri`	a vector containing each number of individual to be deleted. The vector Ri must contain existent individuals, each of them once.

Value

the data frame with transformed variables.

Author(s)

Blondeau Da Silva Stephane - Da Silva Anne.

References

Aitchison J., 1986. The Statistical Analysis of Compositional Data. Chapman and Hall, London - New York, XII, 416 pp.

Peres-Neto P.R., Magnan P., 2004. The influence of swimming demand on phenotypic plasticity and morphological integration: a comparison of two polymorphic charr species. Oecologia 140, 36-45.

Shinn C. et al., 2015. Phenotypic variation as an indicator of pesticide stress in gudgeon: accounting for confounding factors in the wild. Science of the Total Environment 538, 733-742.

Examples

```
data(test)
AitTrans(test)

## The function is currently defined as
function (data, Rp = 0, Ri = 0)
{
    nb.var = dim(data)[2] - 1
    dat.rem.ind.pop <- function(data, ind = 0, pop = 0) {
        data = as.data.frame(data)
        dat.rem.ind <- function(dat, ind) {
            nb.rem.ind = length(ind)
            nb.ind = dim(dat)[1]
            for (i in 1:nb.rem.ind) dat = dat[row.names(dat)[1:(nb.ind -
                i + 1)] != ind[i], ]
            return(dat)
        }
        dat.rem.pop <- function(dat, pop) {
            nb.rem.pop = length(pop)
            for (i in 1:nb.rem.pop) dat = dat[dat[, 1] != pop[i],
                ]
            return(dat)
        }
        if (ind[1] != 0)
            data = dat.rem.ind(data, ind)
        if (pop[1] != 0)
            data = dat.rem.pop(data, pop)
        return(data)
    }
    nonNa.row <- function(data, row) {
        nb.na = 0
        for (i in 1:nb.var) if (is.na(data[row, i + 1]))
            nb.na = nb.na + 1
        return(nb.var - nb.na)
    }
    data = dat.rem.ind.pop(data, ind = Ri, pop = Rp)
    nb.ind = dim(data)[1]
    cor.val = rep(0, nb.ind)
    for (i in 1:nb.ind) {
        cor.val[i] = 1/(nonNa.row(data, i)) * sum(log(data[i,
            -1], base = 10), na.rm = TRUE)
        for (j in 1:nb.var) data[i, (j + 1)] = log(data[i, (j +
            1)], base = 10) - cor.val[i]
    }
    return(data)
  }
```

BootPst *Bootstrap method*

Description

'BootPst' performs a bootstrap resampling procedure with all individuals of the selected populations and calculates Pst values of quantitative measures considered. This function provides a confidence interval or the distribution of Pst.

Usage

```
BootPst(data,va,opt=0,csh=1,boot=1000,Ri=0,Rp=0,Pw=0,pe=0.95,bars=20)
```

Arguments

`data`	a dataframe with as many rows as individuals. The first column contains the name of the population to which the individual belongs, the others contain quantitative variables.
`va`	the name (or number) of the quantitative measure considered.
`opt`	if opt=0 all the boot values of Pst are returned, if opt="ci" these ordered values and the confidence interval are returned, and if opt="hist" these ordered values and the distribution histogram of Pst are returned.
`csh`	the value of c/h^2, where c is the assumed additive genetic proportion of differences between populations and where h^2 is (narrow-sense heritability) the assumed additive genetic proportion of differences between individuals within populations.
`boot`	the number of data frames generated to determine the confidence interval or to construct the distribution (with the bootstrap method).
`Ri`	a vector containing each number of individual to be deleted. The vector Ri must contain existent individuals, each of them once.
`Rp`	a vector containing the names of the populations to be deleted.
`Pw`	a vector containing the names of the two populations considered to obtain pairwise Pst.
`pe`	the confidence level of the calculated interval.
`bars`	the maximum number of bars the histogram may have: indeed, on the x-axis, interval $[0,1]$ is divided into $bars$ parts (there may exist unfilled bars).

Value

In any case, the sizes of each population considered. If opt="ci" an ordered vector containing values of Pst and the confidence interval (also a vector). If opt="hist" an ordered vector containing values of Pst and the Pst distribution histogram. Else a single vector containing the boot values of Pst.

Author(s)

Blondeau Da Silva Stephane - Da Silva Anne.

Examples

```
data(test)
# BootPst(test,va=1)
# BootPst(test,va="QM7",opt="ci",csh=0.8,boot=500,Ri=18)
# BootPst(test,va=11,opt="ci",Ri=c(22,27,195),Rp=c("A","B","E"),pe=0.9)
BootPst(test,va="Body_length",boot=80,opt="hist",bars=50)
# BootPst(test,va=4,opt="hist",Ri=c(3,7:17),Pw=c("C","D"))

## The function is currently defined as
function (data, va, opt = 0, csh = 1, boot = 1000, Ri = 0, Rp = 0,
    Pw = 0, pe = 0.95, bars = 20)
{
    nonNa.clm <- function(data, clm) {
        nb.ind = dim(data)[1]
        nb.na = 0
        for (i in 1:nb.ind) if (is.na(data[i, clm]))
            nb.na = nb.na + 1
        return(nb.ind - nb.na)
    }
    dat.fra.prep <- function(data) {
        nb.var = dim(data)[2] - 1
        data = as.data.frame(data)
        data[, 1] = as.character(data[, 1])
        for (i in 1:nb.var) {
            if (is.numeric(data[, i + 1]) == FALSE)
                data[, i + 1] = as.numeric(as.character(data[,
                  i + 1]))
        }
        dat.sta <- function(dat) {
            nb.vari = dim(dat)[2] - 1
            st.dev = rep(0, nb.vari)
            mea = rep(0, nb.vari)
            for (i in 1:nb.vari) {
                nna.clm = nonNa.clm(dat, i + 1)
                st.dev[i] = sqrt((nna.clm - 1)/nna.clm) * sd(dat[,
                  i + 1], na.rm = TRUE)
                mea[i] = mean(dat[, i + 1], na.rm = TRUE)
            }
            for (j in 1:nb.vari) dat[, j + 1] = (dat[, j + 1] -
                mea[j])/st.dev[j]
            return(dat)
        }
        data = dat.sta(data)
        return(data)
    }
    dat.rem.ind.pop <- function(data, ind = 0, pop = 0) {
        data = as.data.frame(data)
        dat.rem.ind <- function(dat, ind) {
            nb.rem.ind = length(ind)
            nb.ind = dim(dat)[1]
            for (i in 1:nb.rem.ind) dat = dat[row.names(dat)[1:(nb.ind -
                i + 1)] != ind[i], ]
```

```
        return(dat)
    }
    dat.rem.pop <- function(dat, pop) {
        nb.rem.pop = length(pop)
        for (i in 1:nb.rem.pop) dat = dat[dat[, 1] != pop[i],
            ]
        return(dat)
    }
    if (ind[1] != 0)
        data = dat.rem.ind(data, ind)
    if (pop[1] != 0)
        data = dat.rem.pop(data, pop)
    return(data)
}
dat.pw <- function(data, pw = 0) {
    if (pw[1] == 0)
        return(data)
    else {
        data = data[data[, 1] == pw[1] | data[, 1] == pw[2],
            ]
        return(data)
    }
}
nb.pop <- function(data) {
    data = data[order(data[, 1]), ]
    nb.ind = dim(data)[1]
    nb.pop = 1
    for (i in 1:(nb.ind - 1)) if (data[i, 1] != data[i +
        1, 1])
        nb.pop = nb.pop + 1
    return(nb.pop)
}
pop.freq <- function(data) {
    data = data[order(data[, 1]), ]
    nb.ind = dim(data)[1]
    dat.fra = as.data.frame(data)
    nb.pop = 1
    for (i in 1:(nb.ind - 1)) if (data[i, 1] != data[i +
        1, 1])
        nb.pop = nb.pop + 1
    pop.freq.vec = rep(1, nb.pop)
    name = rep(0, nb.pop)
    k = 1
    name[1] = as.character(dat.fra[1, 1])
    for (i in 2:nb.ind) if (dat.fra[i - 1, 1] == dat.fra[i,
        1])
        pop.freq.vec[k] = pop.freq.vec[k] + 1
    else {
        k = k + 1
        name[k] = as.character(dat.fra[i, 1])
    }
    names(pop.freq.vec) = name
    return(pop.freq.vec)
```

```
}
Pst.val <- function(data, csh = 1) {
    nbpop = nb.pop(data)
    nb.var = dim(data)[2] - 1
    data = data[order(data[, 1]), ]
    if (nbpop == 1)
        return(rep(0, nb.var))
    else {
        pop.freq = pop.freq(data)
        Pst.clm <- function(dat, clm) {
            mea = mean(dat[, clm], na.rm = TRUE)
            nna.clm = nonNa.clm(dat, clm)
            SSTotal = (nna.clm - 1) * var(dat[, clm], na.rm = TRUE)
            mea.pop = rep(0, nbpop)
            nna.pop.freq = rep(0, nbpop)
            nna.pop.freq[1] = nonNa.clm(dat[1:(pop.freq[1]),
              ], clm)
            nb.allna.pop = 0
            if (nna.pop.freq[1] == 0)
              nb.allna.pop = 1
            else mea.pop[1] = mean(dat[1:(pop.freq[1]), clm],
              na.rm = TRUE)
            for (i in 2:nbpop) {
              nna.pop.freq[i] = nonNa.clm(dat[(sum(pop.freq[1:(i -
                1)]) + 1):(sum(pop.freq[1:i])), ], clm)
              if (nna.pop.freq[i] != 0)
                mea.pop[i] = mean(dat[(sum(pop.freq[1:(i -
                  1)]) + 1):(sum(pop.freq[1:i])), clm], na.rm = TRUE)
              else nb.allna.pop = nb.allna.pop + 1
            }
            SSBetween = sum(nna.pop.freq * (mea.pop - mea)^2)
            SSWithin = SSTotal - SSBetween
            if ((nna.clm - nbpop + nb.allna.pop) * (nbpop -
              nb.allna.pop - 1) != 0) {
              MSWithin = SSWithin/(nna.clm - nbpop + nb.allna.pop)
              MSBetween = SSBetween/(nbpop - nb.allna.pop -
                1)
              return(csh * MSBetween/(csh * MSBetween + 2 *
                MSWithin))
            }
            else {
              if ((nna.clm - nbpop + nb.allna.pop) == 0)
                return(1)
              else return(0)
            }
        }
        pst.val = rep(0, nb.var)
        for (j in 1:nb.var) pst.val[j] = Pst.clm(data, j +
            1)
        return(pst.val)
    }
}
boot.pst.va <- function(data, csh, boot, clm) {
```

```
    nb.ind = dim(data)[1]
    dat = data[, c(1, clm)]
    boot.val = rep(0, boot)
    for (i in 1:boot) {
        da = dat[sample(1:nb.ind, nb.ind, T), ]
        boot.val[i] = Pst.val(da, csh)
    }
    return(boot.val)
}
ConInt.pst.va <- function(data, csh, boot, clm, per) {
    boot.pst.val = boot.pst.va(data = data, csh = csh, boot = boot,
        clm = clm)
    boot.pst.val = sort(boot.pst.val)
    print(c(boot.pst.val[floor(boot * (1 - per)/2 + 1)],
        boot.pst.val[ceiling(boot * (per + 1)/2)]))
    return(boot.pst.val)
}
dis.pst.va <- function(data, csh, boot, clm, bars) {
    psts.val = boot.pst.va(data = data, csh = csh, boot = boot,
        clm = clm)
    hist(psts.val, breaks = c(0:bars)/bars, xlab = "Pst",
        ylab = "Frequency", main = c("Pst distribution:",
            names(data)[clm]), col = "gray88")
    return(sort(psts.val))
}
for (i in 2:dim(data)[2]) {
    if (names(data)[i] == va)
        va = i - 1
}
if (is.numeric(va) == FALSE)
    return("va value does not exist!")
data = dat.fra.prep(data)
data = dat.rem.ind.pop(data, ind = Ri, pop = Rp)
data = dat.pw(data, pw = Pw)
print("The studied quantitative variable is:")
print(names(data)[va + 1])
print("Populations sizes are:")
print(pop.freq(data))
if (opt != "ci" & opt != "hist") {
    print(paste(boot, "bootstrap values:"))
    return(boot.pst.va(data, csh = csh, boot = boot, clm = va +
        1))
}
if (opt == "ci") {
    print(paste(100 * pe, "% confidence interval determined by",
        boot, "bootstrap values:"))
    return(ConInt.pst.va(data, csh = csh, boot = boot, clm = va +
        1, per = pe))
}
if (opt == "hist") {
    print(paste(boot, "bootstrap values and", "Pst distribution:"))
    dev.new()
    dis.pst.va(data = data, csh = csh, boot = boot, clm = va +
```

```
            1, bars = bars)
    }
  }
```

`Pst` *Pst values and Pst confidence intervals*

Description

'Pst' calculates Pst values of the quantitative measures considered and also their confidence intervals.

Usage

```
Pst(data, ci = 0, csh = 1, va = 0, boot = 1000, Pw = 0, Rp = 0, Ri = 0, pe = 0.95)
```

Arguments

`data`	a dataframe with as many rows as individuals. The first column contains the name of the population to which the individual belongs, the others contain quantitative variables.
`ci`	if ci=1 the confidence intervals are added to Pst values.
`csh`	the value of c/h^2, where c is the assumed additive genetic proportion of differences between populations and where h^2 is (narrow-sense heritability) the assumed additive genetic proportion of differences between individuals within populations.
`va`	a vector containing the selected variables names or numbers (i.e. those of the quantitative measures considered). If va=0 all the variables are selected.
`boot`	the number of data frames generated to determine the confidence interval with the bootstrap method.
`Pw`	a vector containing the names of the two populations considered to obtain pairwise Pst.
`Rp`	a vector containing the names of the populations to be deleted.
`Ri`	a vector containing each number of individual to be deleted. The vector Ri must contain existent individuals, each of them once.
`pe`	the confidence level of the calculated interval.

Value

The sizes of each population considered. Pst values of the selected populations (for quantitative traits considered) and if ci=1 their confidence intervals.

Author(s)

Blondeau Da Silva Stephane - Da Silva Anne.

References

Spitze K., 1993. Population structure in Daphnia obtusa: Quantitative genetic and allozymic variation. Genetics 135: 367-374.

Leinonen T., Cano J.M., Makinen H., Merila J., 2006. Contrasting patterns of body shape and neutral genetic divergence in marine and lake populations of threespine sticklebacks. Journal of Evolutionary Biology 19: 1803-1812.

Brommer J.E., 2011. Whither Pst? The approximation of Qst by Pst in evolutionary and conservation biology. Journal Evolution Biology 24: 1160-1168.

Examples

```
data(test)
Pst(test)
# Pst(test,csh=0.2,ci=1)
Pst(test,va="QM2",ci=1,Rp="D",boot=50)
# Pst(test,va=c(5,8:11),ci=1,boot=2000,Ri=56,Rp="A",pe=0.9)
# Pst(test,ci=1,Ri=c(7,55:59),Pw=c("A","D"))

## The function is currently defined as
function (data, ci = 0, csh = 1, va = 0, boot = 1000, Pw = 0,
    Rp = 0, Ri = 0, pe = 0.95)
{
    nonNa.clm <- function(data, clm) {
        nb.ind = dim(data)[1]
        nb.na = 0
        for (i in 1:nb.ind) if (is.na(data[i, clm]))
            nb.na = nb.na + 1
        return(nb.ind - nb.na)
    }
    dat.fra.prep <- function(data) {
        nb.var = dim(data)[2] - 1
        data = as.data.frame(data)
        data[, 1] = as.character(data[, 1])
        for (i in 1:nb.var) {
            if (is.numeric(data[, i + 1]) == FALSE)
                data[, i + 1] = as.numeric(as.character(data[,
                  i + 1]))
        }
        dat.sta <- function(dat) {
            nb.vari = dim(dat)[2] - 1
            st.dev = rep(0, nb.vari)
            mea = rep(0, nb.vari)
            for (i in 1:nb.vari) {
                nna.clm = nonNa.clm(dat, i + 1)
                st.dev[i] = sqrt((nna.clm - 1)/nna.clm) * sd(dat[,
                  i + 1], na.rm = TRUE)
                mea[i] = mean(dat[, i + 1], na.rm = TRUE)
            }
            for (j in 1:nb.vari) dat[, j + 1] = (dat[, j + 1] -
                mea[j])/st.dev[j]
            return(dat)
```

```
    }
    data = dat.sta(data)
    return(data)
}
dat.rem.ind.pop <- function(data, ind = 0, pop = 0) {
    data = as.data.frame(data)
    dat.rem.ind <- function(dat, ind) {
        nb.rem.ind = length(ind)
        nb.ind = dim(dat)[1]
        for (i in 1:nb.rem.ind) dat = dat[row.names(dat)[1:(nb.ind -
            i + 1)] != ind[i], ]
        return(dat)
    }
    dat.rem.pop <- function(dat, pop) {
        nb.rem.pop = length(pop)
        for (i in 1:nb.rem.pop) dat = dat[dat[, 1] != pop[i],
            ]
        return(dat)
    }
    if (ind[1] != 0)
        data = dat.rem.ind(data, ind)
    if (pop[1] != 0)
        data = dat.rem.pop(data, pop)
    return(data)
}
dat.pw <- function(data, pw = 0) {
    if (pw[1] == 0)
        return(data)
    else {
        data = data[data[, 1] == pw[1] | data[, 1] == pw[2],
            ]
        return(data)
    }
}
nb.pop <- function(data) {
    data = data[order(data[, 1]), ]
    nb.ind = dim(data)[1]
    nb.pop = 1
    for (i in 1:(nb.ind - 1)) if (data[i, 1] != data[i +
        1, 1])
        nb.pop = nb.pop + 1
    return(nb.pop)
}
pop.freq <- function(data) {
    data = data[order(data[, 1]), ]
    nb.ind = dim(data)[1]
    dat.fra = as.data.frame(data)
    nb.pop = 1
    for (i in 1:(nb.ind - 1)) if (data[i, 1] != data[i +
        1, 1])
        nb.pop = nb.pop + 1
    pop.freq.vec = rep(1, nb.pop)
    name = rep(0, nb.pop)
```

```
    k = 1
    name[1] = as.character(dat.fra[1, 1])
    for (i in 2:nb.ind) if (dat.fra[i - 1, 1] == dat.fra[i,
        1])
        pop.freq.vec[k] = pop.freq.vec[k] + 1
    else {
        k = k + 1
        name[k] = as.character(dat.fra[i, 1])
    }
    names(pop.freq.vec) = name
    return(pop.freq.vec)
}
Pst.val <- function(data, csh = 1) {
    nbpop = nb.pop(data)
    nb.var = dim(data)[2] - 1
    data = data[order(data[, 1]), ]
    if (nbpop == 1)
        return(rep(0, nb.var))
    else {
        pop.freq = pop.freq(data)
        Pst.clm <- function(dat, clm) {
            mea = mean(dat[, clm], na.rm = TRUE)
            nna.clm = nonNa.clm(dat, clm)
            SSTotal = (nna.clm - 1) * var(dat[, clm], na.rm = TRUE)
            mea.pop = rep(0, nbpop)
            nna.pop.freq = rep(0, nbpop)
            nna.pop.freq[1] = nonNa.clm(dat[1:(pop.freq[1]),
              ], clm)
            nb.allna.pop = 0
            if (nna.pop.freq[1] == 0)
              nb.allna.pop = 1
            else mea.pop[1] = mean(dat[1:(pop.freq[1]), clm],
              na.rm = TRUE)
            for (i in 2:nbpop) {
              nna.pop.freq[i] = nonNa.clm(dat[(sum(pop.freq[1:(i -
                1)]) + 1):(sum(pop.freq[1:i])), ], clm)
              if (nna.pop.freq[i] != 0)
                mea.pop[i] = mean(dat[(sum(pop.freq[1:(i -
                  1)]) + 1):(sum(pop.freq[1:i])), clm], na.rm = TRUE)
              else nb.allna.pop = nb.allna.pop + 1
            }
            SSBetween = sum(nna.pop.freq * (mea.pop - mea)^2)
            SSWithin = SSTotal - SSBetween
            if ((nna.clm - nbpop + nb.allna.pop) * (nbpop -
              nb.allna.pop - 1) != 0) {
              MSWithin = SSWithin/(nna.clm - nbpop + nb.allna.pop)
              MSBetween = SSBetween/(nbpop - nb.allna.pop -
                1)
              return(csh * MSBetween/(csh * MSBetween + 2 *
                MSWithin))
            }
            else {
              if ((nna.clm - nbpop + nb.allna.pop) == 0)
```

```
                return(1)
              else return(0)
            }
        }
        pst.val = rep(0, nb.var)
        for (j in 1:nb.var) pst.val[j] = Pst.clm(data, j +
            1)
        return(pst.val)
    }
}
boot.pst.va <- function(data, csh, boot, clm) {
    nb.ind = dim(data)[1]
    dat = data[, c(1, clm)]
    boot.val = rep(0, boot)
    for (i in 1:boot) {
        da = dat[sample(1:nb.ind, nb.ind, T), ]
        boot.val[i] = Pst.val(da, csh)
    }
    return(boot.val)
}
ConInt.pst.va <- function(data, csh, boot, clm, per) {
    boot.pst.val = boot.pst.va(data = data, csh = csh, boot = boot,
        clm = clm)
    boot.pst.val = sort(boot.pst.val)
    return(c(boot.pst.val[floor(boot * (1 - per)/2 + 1)],
        boot.pst.val[ceiling(boot * (per + 1)/2)]))
}
if (va[1] == 0) {
    nb.var = dim(data)[2] - 1
    va = 1:nb.var
}
else {
    nb.var = length(va)
    for (i in 1:nb.var) {
        for (j in 2:dim(data)[2]) {
            if (names(data)[j] == va[i])
              va[i] = j - 1
        }
    }
    va = as.numeric(va)
    if (is.na(sum(va)) == TRUE)
        return("va is not valid!")
}
data = dat.fra.prep(data)
data = dat.rem.ind.pop(data, ind = Ri, pop = Rp)
data = dat.pw(data, Pw)
print("Populations sizes are:")
print(pop.freq(data))
if (ci != 1) {
    output = data.frame(Quant_Varia = names(data)[va + 1],
        Pst_Values = Pst.val(data, csh = csh)[va], row.names = NULL)
    return(output)
}
```

```
  if (ci == 1) {
      low.bnd = rep(0, nb.var)
      up.bnd = rep(0, nb.var)
      for (i in 1:nb.var) {
          ci.pst.va = ConInt.pst.va(data, csh = csh, boot = boot,
              clm = va[i] + 1, per = pe)
          low.bnd[i] = ci.pst.va[1]
          up.bnd[i] = ci.pst.va[2]
      }
      output = data.frame(Quant_Varia = names(data)[va + 1],
          Pst_Values = Pst.val(data, csh = csh)[va], LowBoundCI = low.bnd,
          UpBoundCI = up.bnd, row.names = NULL)
      names(output)[3] = paste(100 * pe, "%_LowBoundCI")
      names(output)[4] = paste(100 * pe, "%_UpBoundCI")
      return(output)
  }
}
```

ReistTrans *Reist standardization*

Description

'ReistTrans' calculates residuals (size adjusted measurements) from Reist tranformations to eliminate any variation resulting from allometric growth. There is a single regressor (one of the quantitative traits).

Usage

```
ReistTrans(data, reg, Rp = 0, Ri = 0)
```

Arguments

data	a dataframe with as many rows as individuals. The first column contains the name of the population to which the individual belongs, the others contain quantitative variables.
reg	the name (or the rank) of the variable chosen as the explanatory variable.
Rp	a vector containing the names of the populations to be deleted.
Ri	a vector containing each number of individual to be deleted. The vector Ri must contain existent individuals, each of them once.

Value

the data frame of adjusted variables, the column containing the quantitative trait used as a regressor being deleted.

Note

dispensable quantitative measures can easily be deleted in the main functions of R.

Author(s)

Blondeau Da Silva Stephane - Da Silva Anne.

References

Reist J.D., 1985. An empirical evaluation of several univariate methods that adjust for size variation in morphometric data. Canadian Journal Zoology 63, 1429-1439.

Kaeuffer R. et al., 2012. Parallel and nonparallel aspects of ecological, phenotypic, and genetic divergence across replicate population pairs of lake and stream stickleback. Evolution 66(2), 402-418.

He Y. et al., 2013. Morphological Variation Among Wild Populations of Chinese Rare Minnow (Gobiocypris rarus): Deciphering the Role of Evolutionary Processes. Zoological Science 30, 475-483.

Examples

```
data(test)
names(test)[9]
ReistTrans(test,reg=9)

## The function is currently defined as
function (data, reg, Rp = 0, Ri = 0)
{
    dat.rem.ind.pop <- function(data, ind = 0, pop = 0) {
        data = as.data.frame(data)
        dat.rem.ind <- function(dat, ind) {
            nb.rem.ind = length(ind)
            nb.ind = dim(dat)[1]
            for (i in 1:nb.rem.ind) dat = dat[row.names(dat)[1:(nb.ind -
                i + 1)] != ind[i], ]
            return(dat)
        }
        dat.rem.pop <- function(dat, pop) {
            nb.rem.pop = length(pop)
            for (i in 1:nb.rem.pop) dat = dat[dat[, 1] != pop[i],
                ]
            return(dat)
        }
        if (ind[1] != 0)
            data = dat.rem.ind(data, ind)
        if (pop[1] != 0)
            data = dat.rem.pop(data, pop)
        return(data)
    }
    Reitra.va <- function(dat, clm, re) {
        dat = dat[is.finite(dat[, re]), ]
        log.dat = dat
        mea = mean(dat[is.finite(dat[, clm]), re])
        log.dat[, clm] = log(dat[, clm], base = 10)
        log.dat[, re] = log(dat[, re], base = 10)
        mea.clm = mean(log.dat[is.finite(log.dat[, clm]), clm],
```

```
            na.rm = TRUE)
        mea.reg = mean(log.dat[is.finite(log.dat[, clm]), re],
            na.rm = TRUE)
        a = sum((log.dat[is.finite(log.dat[, clm]), re] - mea.reg) *
            log.dat[is.finite(log.dat[, clm]), clm])/sum((log.dat[is.finite(log.dat[,
            clm]), re] - mea.reg) * (log.dat[is.finite(log.dat[,
            clm]), re] - mea.reg))
        dat[, clm] = log.dat[, clm] - a * (log.dat[, re] - log(mea,
            base = 10))
        return(dat)
    }
    nb.var = dim(data)[2] - 1
    for (i in 1:nb.var) {
        if (names(data)[i + 1] == reg)
            reg = i
    }
    if (is.numeric(reg) == FALSE)
        return("reg value does not exist!")
    data = dat.rem.ind.pop(data, ind = Ri, pop = Rp)
    if (reg == 1)
        for (i in 2:nb.var) data = Reitra.va(data, clm = i +
            1, re = 2)
    else {
        for (i in 2:reg) data = Reitra.va(data, clm = i, re = reg +
            1)
        if (reg < nb.var)
            for (j in (reg + 1):nb.var) data = Reitra.va(data,
                clm = j + 1, re = reg + 1)
    }
    return(data[-(reg + 1)])
}
```

`Res` *Residuals from a linear regression*

Description

'Res' calculates residuals from simple linear regressions (in particular to eliminate any variation resulting from allometric growth). These regression adjustments assume the existence of linear relationships between the dependent variables and the regressor (one of the column of the data frame).

Usage

```
Res(data, reg, Rp = 0, Ri = 0)
```

Arguments

`data`	a dataframe with as many rows as individuals. The first column contains the name of the population to which the individual belongs, the others contain quantitative variables.

reg	the name (or the rank) of the variable chosen as the explanatory variable.
Rp	a vector containing the names of the populations to be deleted.
Ri	a vector containing each number of individual to be deleted. The vector Ri must contain existent individuals, each of them once.

Value

the data frame of adjusted variables, the column containing the quantitative trait used as a regressor being deleted.

Note

dispensable quantitative measures can easily be deleted in the main functions of R.

Author(s)

Blondeau Da Silva Stephane - Da Silva Anne.

Examples

```
# data(test)
# names(test)[9]
# Res(test,r=9)
## The function is currently defined as
function (data, reg, Rp = 0, Ri = 0)
{
    dat.rem.ind.pop <- function(data, ind = 0, pop = 0) {
        data = as.data.frame(data)
        dat.rem.ind <- function(dat, ind) {
            nb.rem.ind = length(ind)
            nb.ind = dim(dat)[1]
            for (i in 1:nb.rem.ind) dat = dat[row.names(dat)[1:(nb.ind -
                i + 1)] != ind[i], ]
            return(dat)
        }
        dat.rem.pop <- function(dat, pop) {
            nb.rem.pop = length(pop)
            for (i in 1:nb.rem.pop) dat = dat[dat[, 1] != pop[i],
                ]
            return(dat)
        }
        if (ind[1] != 0)
            data = dat.rem.ind(data, ind)
        if (pop[1] != 0)
            data = dat.rem.pop(data, pop)
        return(data)
    }
    Res.va <- function(dat, clm, re) {
        dat = dat[is.finite(dat[, re]), ]
        mea.clm = mean(dat[is.finite(dat[, clm]), clm], na.rm = TRUE)
        mea.reg = mean(dat[is.finite(dat[, clm]), re], na.rm = TRUE)
```

```
        a = sum((dat[is.finite(dat[, clm]), re] - mea.reg) *
            dat[is.finite(dat[, clm]), clm])/sum((dat[is.finite(dat[,
            clm]), re] - mea.reg) * (dat[is.finite(dat[, clm]),
            re] - mea.reg))
        b = mea.clm - a * mea.reg
        dat[, clm] = dat[, clm] - b - a * dat[, re]
        return(dat)
    }
    nb.var = dim(data)[2] - 1
    for (i in 1:nb.var) {
        if (names(data)[i + 1] == reg)
            reg = i
    }
    if (is.numeric(reg) == FALSE)
        return("reg value does not exist!")
    data = dat.rem.ind.pop(data, ind = Ri, pop = Rp)
    if (reg == 1)
        for (i in 2:nb.var) data = Res.va(data, clm = i + 1,
            re = 2)
    else {
        for (i in 2:reg) data = Res.va(data, clm = i, re = reg +
            1)
        if (reg < nb.var)
            for (j in (reg + 1):nb.var) data = Res.va(data, clm = j +
                1, re = reg + 1)
    }
    return(data[-(reg + 1)])
}
```

`test` *Example of a data frame*

Description

a dataframe containing 200 rows (i.e. 200 individuals). The first column contains the name of the population ("A", "B", "C", "D" or "E") to which the individual belongs, the 11 others contain quantitative measures.

Usage

```
data("test")
```

References

Michalski S.G., Durka W. 2015. Data from: Separation in flowering time contributes to the maintenance of sympatric cryptic plant lineages. Ecology and Evolution 5 (11): 2172-2184. Dryad Digital Repository. http://dx.doi.org/10.5061/dryad.bk5hk.

Examples

```
data(test)
```

`TracePst` *Pst variations in function of c/h^2*

Description

'TracePst' plots the curves of the functions that map c/h^2 onto Pst (for chosen quantitative measures). Indeed, Pst depends on the value of c/h^2, where c is the assumed additive genetic proportion of differences between populations and where h^2 is (narrow-sense heritability) the assumed additive genetic proportion of differences between individuals within populations.

Usage

```
TracePst(data,va=0,ci=1,boot=1000,pe=0.95,Fst=-1,Pw=0,Rp=0,Ri=0,xm=2,pts=30)
```

Arguments

`data`	a dataframe with as many rows as individuals. The first column contains the name of the population to which the individual belongs, the others contain quantitative variables.
`va`	a vector containing the selected variables names or numbers (i.e. those of the quantitative measures considered). If va=0 all the variables are selected.
`ci`	if ci=1 the confidence interval of Pst is plotted.
`boot`	the number of data frames generated to determine the confidence interval or to construct the dotted lines representing this confidence interval (using the bootstrap method).
`pe`	the confidence level of the calculated interval.
`Fst`	the value of Wright's Fst, if avalaible.
`Pw`	a vector containing the names of the two populations considered to obtain pairwise Pst.
`Rp`	a vector containing the names of the populations to be deleted.
`Ri`	a vector containing each number of individual to be deleted. The vector Ri must contain existent individuals, each of them once.
`xm`	the maximum on x-axis (values of c/h^2).
`pts`	number of points used to plot the curves.

Value

In any case, the sizes of each population considered. The expected curves.

Note

The time required to construct the dotted lines associated with the confidence intervals might be fairly long depending on the user choices.

Author(s)

Blondeau Da Silva Stephane - Da Silva Anne.

References

Brommer J., 2011. Whither Pst? The approximation of Qst by Pst in evolutionary and conservation biology. Journal of Evolutionary Biology, 24:1160-1168.

Lima M.R. et al., 2012. Genetic and Morphometric Divergence of an Invasive Bird: The Introduced House Sparrow (Passer domesticus) in Brazil. PloS One 7 (12).

On Fst : Wright S., 1951. The genetical structure of populations. Annals of Eugenics 15, 323-354.

Examples

```
data(test)
# TracePst(test)
# TracePst(test,boot=2000,va="QM7",Ri=18,pe=0.9,pts=40,xm=4)
TracePst(test,va=7:10,Fst=0.3,Ri=c(22,27,195),Rp=c("A","C","E"),ci=0)
# TracePst(test,va="QM1",Ri=c(3,7:17),Pw=c("C","D"),pts=20)

## The function is currently defined as
function (data, va = 0, ci = 1, boot = 1000, pe = 0.95, Fst = -1,
    Pw = 0, Rp = 0, Ri = 0, xm = 2, pts = 30)
{
    nonNa.clm <- function(data, clm) {
        nb.ind = dim(data)[1]
        nb.na = 0
        for (i in 1:nb.ind) if (is.na(data[i, clm]))
            nb.na = nb.na + 1
        return(nb.ind - nb.na)
    }
    dat.fra.prep <- function(data) {
        nb.var = dim(data)[2] - 1
        data = as.data.frame(data)
        data[, 1] = as.character(data[, 1])
        for (i in 1:nb.var) {
            if (is.numeric(data[, i + 1]) == FALSE)
                data[, i + 1] = as.numeric(as.character(data[,
                  i + 1]))
        }
        dat.sta <- function(dat) {
            nb.vari = dim(dat)[2] - 1
            st.dev = rep(0, nb.vari)
            mea = rep(0, nb.vari)
            for (i in 1:nb.vari) {
                nna.clm = nonNa.clm(dat, i + 1)
                st.dev[i] = sqrt((nna.clm - 1)/nna.clm) * sd(dat[,
```

```
            i + 1], na.rm = TRUE)
        mea[i] = mean(dat[, i + 1], na.rm = TRUE)
    }
    for (j in 1:nb.vari) dat[, j + 1] = (dat[, j + 1] -
        mea[j])/st.dev[j]
    return(dat)
}
data = dat.sta(data)
return(data)
}
dat.rem.ind.pop <- function(data, ind = 0, pop = 0) {
    data = as.data.frame(data)
    dat.rem.ind <- function(dat, ind) {
        nb.rem.ind = length(ind)
        nb.ind = dim(dat)[1]
        for (i in 1:nb.rem.ind) dat = dat[row.names(dat)[1:(nb.ind -
            i + 1)] != ind[i], ]
        return(dat)
    }
    dat.rem.pop <- function(dat, pop) {
        nb.rem.pop = length(pop)
        for (i in 1:nb.rem.pop) dat = dat[dat[, 1] != pop[i],
            ]
        return(dat)
    }
    if (ind[1] != 0)
        data = dat.rem.ind(data, ind)
    if (pop[1] != 0)
        data = dat.rem.pop(data, pop)
    return(data)
}
dat.pw <- function(data, pw = 0) {
    if (pw[1] == 0)
        return(data)
    else {
        data = data[data[, 1] == pw[1] | data[, 1] == pw[2],
            ]
        return(data)
    }
}
nb.pop <- function(data) {
    data = data[order(data[, 1]), ]
    nb.ind = dim(data)[1]
    nb.pop = 1
    for (i in 1:(nb.ind - 1)) if (data[i, 1] != data[i +
        1, 1])
        nb.pop = nb.pop + 1
    return(nb.pop)
}
pop.freq <- function(data) {
    data = data[order(data[, 1]), ]
    nb.ind = dim(data)[1]
    dat.fra = as.data.frame(data)
```

```
    nb.pop = 1
    for (i in 1:(nb.ind - 1)) if (data[i, 1] != data[i +
        1, 1])
        nb.pop = nb.pop + 1
    pop.freq.vec = rep(1, nb.pop)
    name = rep(0, nb.pop)
    k = 1
    name[1] = as.character(dat.fra[1, 1])
    for (i in 2:nb.ind) if (dat.fra[i - 1, 1] == dat.fra[i,
        1])
        pop.freq.vec[k] = pop.freq.vec[k] + 1
    else {
        k = k + 1
        name[k] = as.character(dat.fra[i, 1])
    }
    names(pop.freq.vec) = name
    return(pop.freq.vec)
}
Pst.val <- function(data, csh = 1) {
    nbpop = nb.pop(data)
    nb.var = dim(data)[2] - 1
    data = data[order(data[, 1]), ]
    if (nbpop == 1)
        return(rep(0, nb.var))
    else {
        pop.freq = pop.freq(data)
        Pst.clm <- function(dat, clm) {
            mea = mean(dat[, clm], na.rm = TRUE)
            nna.clm = nonNa.clm(dat, clm)
            SSTotal = (nna.clm - 1) * var(dat[, clm], na.rm = TRUE)
            mea.pop = rep(0, nbpop)
            nna.pop.freq = rep(0, nbpop)
            nna.pop.freq[1] = nonNa.clm(dat[1:(pop.freq[1]),
              ], clm)
            nb.allna.pop = 0
            if (nna.pop.freq[1] == 0)
              nb.allna.pop = 1
            else mea.pop[1] = mean(dat[1:(pop.freq[1]), clm],
              na.rm = TRUE)
            for (i in 2:nbpop) {
              nna.pop.freq[i] = nonNa.clm(dat[(sum(pop.freq[1:(i -
                1)]) + 1):(sum(pop.freq[1:i])), ], clm)
              if (nna.pop.freq[i] != 0)
                mea.pop[i] = mean(dat[(sum(pop.freq[1:(i -
                  1)]) + 1):(sum(pop.freq[1:i])), clm], na.rm = TRUE)
              else nb.allna.pop = nb.allna.pop + 1
            }
            SSBetween = sum(nna.pop.freq * (mea.pop - mea)^2)
            SSWithin = SSTotal - SSBetween
            if ((nna.clm - nbpop + nb.allna.pop) * (nbpop -
              nb.allna.pop - 1) != 0) {
              MSWithin = SSWithin/(nna.clm - nbpop + nb.allna.pop)
              MSBetween = SSBetween/(nbpop - nb.allna.pop -
```

```
                1)
              return(csh * MSBetween/(csh * MSBetween + 2 *
                MSWithin))
            }
            else {
              if ((nna.clm - nbpop + nb.allna.pop) == 0)
                return(1)
              else return(0)
            }
        }
        pst.val = rep(0, nb.var)
        for (j in 1:nb.var) pst.val[j] = Pst.clm(data, j +
            1)
        return(pst.val)
    }
}
boot.pst.va <- function(data, csh, boot, clm) {
    nb.ind = dim(data)[1]
    dat = data[, c(1, clm)]
    boot.val = rep(0, boot)
    for (i in 1:boot) {
        da = dat[sample(1:nb.ind, nb.ind, T), ]
        boot.val[i] = Pst.val(da, csh)
    }
    return(boot.val)
}
ConInt.pst.va <- function(data, csh, boot, clm, per) {
    boot.pst.val = boot.pst.va(data = data, csh = csh, boot = boot,
        clm = clm)
    boot.pst.val = sort(boot.pst.val)
    return(c(boot.pst.val[floor(boot * (1 - per)/2 + 1)],
        boot.pst.val[ceiling(boot * (per + 1)/2)]))
}
Trace <- function(data, pts, boot, Fst, xm, ci) {
    tra.pst.fst.va <- function(data, pts, clm, Fst, xmax,
        lab.pos) {
        data = data[, c(1, clm)]
        points <- function(nb.pts) {
            pst.va = Pst.val(data, 0)
            for (i in 1:nb.pts) pst.va = c(pst.va, Pst.val(data,
              xmax * i/nb.pts))
            return(pst.va)
        }
        pst.val = points(nb.pts = pts)
        csh.val = xm * c(0:pts)/pts
        plot(pst.val ~ csh.val, type = "l", xlab = "c/h^2",
            ylab = "Pst", main = c("Pst variations:", names(data)[2]),
            ylim = c(0, 1), col = "firebrick1")
        if (Fst != -1) {
            abline(h = Fst, col = "green", lty = 4)
            text(0.05 * lab.pos - 0.06, Fst + 0.04 * lab.pos -
              0.01, "Fst", col = "green")
        }
```

```
    }
    tra.confint.va <- function(clm) {
        point <- function(nb.pt) {
            ci.pst.va = ConInt.pst.va(data, csh = 0, boot = boot,
              clm = clm, per = pe)
            upbnd.val = ci.pst.va[2]
            lowbnd.val = ci.pst.va[1]
            for (i in 1:nb.pt) ci.pst.va = c(ci.pst.va, ConInt.pst.va(data,
              csh = xm * i/nb.pt, boot = boot, clm = clm,
              per = pe))
            for (i in 1:nb.pt) upbnd.val = c(ci.pst.va[2 +
              2 * i], upbnd.val)
            for (i in 1:nb.pt) lowbnd.val = c(lowbnd.val,
              ci.pst.va[1 + 2 * i])
            return(c(upbnd.val, lowbnd.val))
        }
        ci.pst.val = point(nb.pt = pts)
        csh.val = xm * c(0:pts)/pts
        rev.csh.val = rev(csh.val)
        plot(ci.pst.val ~ c(rev.csh.val, csh.val), type = "l",
            xlab = "c/h^2", ylab = "Pst", main = c("Pst variations:",
              names(data)[clm]), ylim = c(0, 1), col = "chocolate4",
            lty = 2)
    }
    nb.var = dim(data)[2] - 1
    nb.gra.lon = ceiling(sqrt(nb.var))
    par(mfrow = c(nb.gra.lon, nb.gra.lon))
    for (i in 1:nb.var) {
        tra.pst.fst.va(data, pts = pts, Fst = Fst, clm = i +
            1, xmax = xm, lab.pos = nb.gra.lon)
        if (ci == 1) {
            par(new = TRUE)
            tra.confint.va(clm = i + 1)
        }
    }
}
if (va[1] == 0) {
    nb.var = dim(data)[2] - 1
    va = 1:nb.var
}
else {
    nb.var = length(va)
    for (i in 1:nb.var) {
        for (j in 2:dim(data)[2]) {
            if (names(data)[j] == va[i])
              va[i] = j - 1
        }
    }
    va = as.numeric(va)
    if (is.na(sum(va)) == TRUE)
        return("va is not valid!")
}
data = dat.fra.prep(data)
```

```
  data = dat.rem.ind.pop(data, ind = Ri, pop = Rp)
  data = dat.pw(data, Pw)
  print("Populations sizes are:")
  print(pop.freq(data))
  data = data[, c(1, va + 1)]
  dev.new()
  Trace(data, pts = pts, boot = boot, Fst = Fst, xm = xm, ci = ci)
}
```

Index

AitTrans, 3

BootPst, 4

Pst, 10
Pstat *(*Pstat-package*)*, 2
Pstat-package, 2

ReistTrans, 15
Res, 17

test, 19
TracePst, 20