

Package ‘aqp’

January 3, 2019

Version 1.17

Date 2018-12-30

Title Algorithms for Quantitative Pedology

Author Dylan Beaudette [aut, cre],
Pierre Roudier [aut, ctb],
Andrew Brown [aut, ctb]

Maintainer Dylan Beaudette <dylan.beaudette@ca.usda.gov>

Depends R (>= 3.0.0)

Imports grDevices, graphics, stats, utils, methods, plyr, reshape,
grid, lattice, cluster, sp, Hmisc, stringr, scales, plotrix,
RColorBrewer, MASS, digest

Suggests colorspace, maptools, foreign, ape, soilDB, latticeExtra,
maps, compositions, sharpshootR, markovchain, xtable, testthat,
Gmedian, farver

Description The Algorithms for Quantitative Pedology (AQP) project was started in 2009 to organize a loosely-related set of concepts and source code on the topic of soil profile visualization, aggregation, and classification into this package (aqp). Over the past 8 years, the project has grown into a suite of related R packages that enhance and simplify the quantitative analysis of soil profile data. Central to the AQP project is a new vocabulary of specialized functions and data structures that can accommodate the inherent complexity of soil profile information; freeing the scientist to focus on ideas rather than boilerplate data processing tasks <doi:10.1016/j.cageo.2012.10.020>. These functions and data structures have been extensively tested and documented, applied to projects involving hundreds of thousands of soil profiles, and deeply integrated into widely used tools such as Soil-Web <<https://casoilresource.lawr.ucdavis.edu/soilweb-apps/>>. Components of the AQP project (aqp, soilDB, sharpshootR, soilReports packages) serve an important role in routine data analysis within the USDA-NRCS Soil Science Division. The AQP suite of R packages offer a convenient platform for bridging the gap between pedometric theory and practice.

License GPL (>= 3)

LazyLoad yes

Repository CRAN

URL <https://github.com/ncss-tech/aqp>

BugReports <https://github.com/ncss-tech/aqp/issues>

Encoding UTF-8

RoxygenNote 6.1.0

NeedsCompilation no

Date/Publication 2019-01-03 06:50:03 UTC

R topics documented:

aqp-package	3
addBracket	4
addVolumeFraction	6
aggregateColor	7
aggregateSoilDepth	9
argillic.clay.increase.depth	10
brierScore	11
ca630	12
checkSPC	15
clod.hz.ids	16
colorQuantiles	17
confusionIndex	18
crit.clay.argillic	19
estimatePSCS	20
estimateSoilDepth	22
evalGenHZ	23
evalMissingData	25
f.noise	26
generalize.hz	29
get.increase.depths	30
get.increase.matrix	31
get.ml.hz	32
getArgillicBounds	34
getSoilDepthClass	35
getSurfaceHorizonDepth	36
glom	38
groupedProfilePlot	39
guessGenHzLevels	41
hzDistinctnessCodeToOffset	43
hzTransitionProbabilities	44
missingDataGrid	46
munsell	47
munsell2rgb	48
panel.depth_function	51
plotMultipleSPC	53
plot_distance_graph	54
previewColors	55

profileApply-methods	57
profileGroupLabels	60
profile_compare-methods	62
random_profile	66
rebuildSPC	69
resample.twotheta	69
rruff.sample	71
shannonEntropy	72
sim	73
slab-methods	74
slice-methods	81
soilColorSignature	83
SoilProfileCollection-class	85
SoilProfileCollection-plotting-methods	87
soil_minerals	90
sp1	92
sp2	94
sp3	96
sp4	98
sp5	101
sp6	105
SPC-utils	106
subsetProfiles-methods	107
tau	108
test_hz_logic	110
textureTriangleSummary	112
union	114
unique-methods	116
unroll	117
Index	119

Description

The aqp (Algorithms for Quantitative Pedology) package for R was developed to address some of the difficulties associated with processing soils information, specifically related to visualization, aggregation, and classification of soil profile data. This package is based on a mix of S3/S4 functions and classes, and most functions use basic dataframes as input, where rows represent soil horizons and columns define properties of those horizons. Common to most functions are the requirements that horizon boundaries are defined as depth from 0, and that profiles are uniquely defined by an id column. The aqp package defines an S4 class, "SoilProfileCollection", for storage of profile-level metadata, as well as summary, print, and plotting methods that have been customized for common tasks related to soils data.

Demos: `demo(aqp)`

```
demo(slope_effect_hz_thickness)
```

[Project homepage.](#)

Author(s)

Dylan E. Beaudette <debeaudette@ucdavis.edu> and Pierre Roudier

See Also

[ca630](#), [sp1](#), [sp2](#), [sp3](#), [sp4](#), [sp5](#)

addBracket

Add Depth Brackets

Description

Add depth-wise brackets to an existing plot of a SoilProfileCollection object.

Usage

```
addBracket(x, label.cex=0.75,
  tick.length = 0.05, arrow.length = 0.05, offset = -0.3,
  missing.bottom.depth = NULL,
  ...)
```

Arguments

x	data.frame containing ID, 'top', 'bottom', and optionally 'label' columns
label.cex	scaling factor for label font
tick.length	length of bracket "tick" mark
arrow.length	length of arrowhead
offset	numeric, length of left-hand offset from each profile
missing.bottom.depth	distance (in depth units) to extend brackets that are missing a lower depth (defaults to max depth of collection)
...	further arguments passed on to segments or arrows

Details

Additional examples can be found in [this tutorial](#).

Note

This is a 'low-level' plotting function: you must first plot a SoilProfileCollection object before using this function.

Author(s)

D.E. Beaudette

See Also[plotSPC](#)**Examples**

```
library(plyr)

# sample data
data(sp1)

# add color vector
sp1$soil_color <- with(sp1, munsell2rgb(hue, value, chroma))

# promote to SoilProfileCollection
depths(sp1) <- id ~ top + bottom

# plot profiles
plot(sp1)

# extract top/bottom depths associated with all A horizons
# return as a single data.frame / profile
f <- function(i) {
  h <- horizons(i)
  idx <- grep('^A', h$name)
  res <- data.frame(top=min(h$top[idx]), bottom=max(h$bottom[idx], na.rm=TRUE))
  return(res)
}

# apply function to each profile in sp1, result is a list
a <- profileApply(sp1, f, simplify=FALSE)
# convert list into data.frame
a <- ldply(a)
# set idname so that addBrackets can locate the correct profile
names(a)[1] <- idname(sp1)

# plot
plot(sp1)
# annotate with brackets
# note that plotting order is derived from the call to `plot(sp1)`
addBracket(a, col='red')

# brackets follow plotting order
plot(sp1, plot.order=length(sp1):1)
# annotate with brackets
# note that plotting order is derived from the call to `plot(sp1)`
addBracket(a, col='red')
```

addVolumeFraction *Symbolize Volume Fraction on a Soil Profile Collection Plot*

Description

Symbolize volume fraction on an existing soil profile collection plot.

Usage

```
addVolumeFraction(x, colname, res = 10, cex.min = 0.1,  
cex.max = 0.5, pch = 1, col = "black")
```

Arguments

x	a SoilProfileCollection object
colname	character vector of length 1, naming the column containing volume fraction data (horizon-level attribute)
res	integer, resolution of the grid used to symbolize volume fraction
cex.min	minimum symbol size
cex.max	maximum symbol size
pch	plotting character
col	color of the symbol

Details

This function can only be called after plotting a SoilProfileCollection object.

Note

Details associated with a call to plot.SoilProfileCollection are automatically accounted for within this function: e.g. plot.order, width, etc.

Author(s)

D.E. Beaudette

See Also

[plotSPC](#)

Examples

```

# sample data
data(loafercreek, package='soilDB')

# subset first 10 profiles
s <- loafercreek[1:10, ]

# replace 0% frags with NA
s$total_frags_pct[which(s$total_frags_pct == 0)] <- NA

# plot in random order, note that annotations follow
par(mar=c(0, 0, 3, 0))
plot(s, color='total_frags_pct', plot.order=sample(1:length(s)))
addVolumeFraction(s, 'total_frags_pct', pch=1)

par(mar=c(0, 0, 0, 0))
plot(s, max.depth=100, name='total_frags_pct', cex.name=1, axis.line.offset=-4.25)
addVolumeFraction(s, 'total_frags_pct', pch=1)

```

aggregateColor

Summarize Soil Colors

Description

Summarize soil color data, weighted by occurrence and horizon thickness.

Usage

```
aggregateColor(x, groups = "genhz", col = "soil_color", k=NULL)
```

Arguments

x	a SoilProfileCollection object
groups	the name of a horizon or site attribute used to group horizons, see examples
col	the name of a horizon-level attribute with soil color specified in hexadecimal (i.e. "#rrggbb")
k	single integer specifying the number of colors discretized via PAM, see details

Details

Weights are computed by: $w_i = \sqrt{\text{sum}(\text{thickness}_i)} * n_i$ where w_i is the weight associated with color i , thickness_i is the total thickness of all horizons associated with the color i , and n_i is the number of horizons associated with color i . Weights are computed within groups specified by groups.

Value

A list with the following components:

scaled.data a list of colors and associated weights, one item for each generalized horizon label with at least one color specified in the source data

aggregate.data a data.frame of weighted-mean colors, one row for each generalized horizon label with at least one color specified in the source data

Author(s)

D.E. Beaudette

See Also

[generalize.hz](#)

Examples

```
# load some example data
data(sp1, package='aqp')

# upgrade to SoilProfileCollection and convert Munsell colors
sp1$soil_color <- with(sp1, munsell2rgb(hue, value, chroma))
depths(sp1) <- id ~ top + bottom
site(sp1) <- ~ group

# generalize horizon names
n <- c('O', 'A', 'B', 'C')
p <- c('O', 'A', 'B', 'C')
sp1$genhz <- generalize.hz(sp1$name, n, p)

# aggregate colors over horizon-level attribute: 'genhz'
a <- aggregateColor(sp1, groups = 'genhz', col = 'soil_color')

# aggregate colors over site-level attribute: 'group'
a <- aggregateColor(sp1, groups = 'group', col = 'soil_color')

# aggregate colors over site-level attribute: 'group'
# discretize colors to 4 per group
a <- aggregateColor(sp1, groups = 'group', col = 'soil_color', k = 4)

# aggregate colors over depth-slices
s <- slice(sp1, c(5, 10, 15, 25, 50, 100, 150) ~ soil_color)
s$slice <- paste0(s$top, ' cm')
s$slice <- factor(s$slice, levels=guessGenHzLevels(s, 'slice')$levels)
a <- aggregateColor(s, groups = 'slice', col = 'soil_color')

## Not run:
# optionally plot with helper function
if(require(sharpshootR))
  aggregateColorPlot(a)
```

```
## End(Not run)

# a more interesting example
## Not run:
data(loafercreek, package = 'soilDB')

# generalize horizon names using REGEX rules
n <- c('Oi', 'A', 'BA', 'Bt1', 'Bt2', 'Bt3', 'Cr', 'R')
p <- c('O', '^A$|Ad|Ap|AB', 'BA$|Bw',
      'Bt1$|^B$', '^Bt$|^Bt2$', '^Bt3|^Bt4|CBt$|BCt$|2Bt|2CB$|^C$', 'Cr', 'R')
loafercreek$genhz <- generalize.hz(loafercreek$hznname, n, p)

# remove non-matching generalized horizon names
loafercreek$genhz[loafercreek$genhz == 'not-used'] <- NA
loafercreek$genhz <- factor(loafercreek$genhz)

a <- aggregateColor(loafercreek, 'genhz')

# plot results with helper function
par(mar=c(1,4,4,1))
aggregateColorPlot(a, print.n.hz = TRUE)

# inspect aggregate data
a$aggregate.data

## End(Not run)
```

aggregateSoilDepth *Probabilistic Estimation of Soil Depth*

Description

Estimate the most-likely depth to contact within a collection of soil profiles.

Usage

```
aggregateSoilDepth(x, groups, crit.prob = 0.9, name = "hznname", p = "Cr|R|Cd", ...)
```

Arguments

x	a SoilProfileCollection object
groups	the name of a site-level attribute that defines groups of profiles within a collection
crit.prob	probability cutoff used to determine where the most likely depth to contact will be, e.g. 0.9 translates to 90% of profiles are shallower than this depth
name	horizon-level attribute where horizon designation is stored
p	a REGEX pattern that matches non-soil genetic horizons
...	additional arguments to slab

Details

This function computes a probability-based estimate of soil depth by group. If no grouping variable exists, a dummy value can be used to compute a single estimate. The `crit.prob` argument sets the critical probability (e.g. 0.9) at which soil depth within a group of profiles is determined. For example, a `crit.prob` of 0.95 might result in an estimated soil depth (e.g. 120cm) where 95% of the profiles (by group) had depths that were less than or equal to 120cm.

Value

A `data.frame` is returned, with as many rows as there are unique group labels, as specified in `groups`.

Author(s)

D.E. Beaudette

See Also

[estimateSoilDepth,slab](#)

Examples

```
data(sp1)
depths(sp1) <- id ~ top + bottom
site(sp1) <- ~ group

aggregateSoilDepth(sp1, 'group', crit.prob = 0.9, name='name')
```

argillic.clay.increase.depth

Return upper boundary of argillic horizon

Description

Returns the top depth of the argillic horizon as a numeric vector.

Uses `crit.clay.argillic` to determine threshold clay increase, and `get.increase.matrix` to determine where increase is met within a vertical distance of 30 cm.

Usage

```
argillic.clay.increase.depth(p, attr = "clay")
```

Arguments

`p` A single-profile `SoilProfileCollection` object.
`attr` OPTIONAL: horizon attribute name referring to clay content. default: 'clay'

Value

A numeric vector containing top depth of argillic horizon, if present, or NA.

Author(s)

Andrew Gene Brown

See Also

`getArgillicBounds`, `get.increase.matrix`, `crit.clay.argillic`

Examples

```
data(sp1, package = 'aqp')
depths(sp1) <- id ~ top + bottom
site(sp1) <- ~ group

p <- sp1[[1]]
attr <- 'prop' # clay contents
foo <- argillic.clay.increase.depth(p, attr = attr)
foo
```

brierScore

Multinomial Brier score

Description

Compute the multinomial Brier score

Usage

```
brierScore(x, classLabels, actual = "actual")
```

Arguments

<code>x</code>	data.frame, rows are predictions/observations, columns contain classes
<code>classLabels</code>	vector of predicted class labels (probabilities), corresponding to column names in <code>x</code>
<code>actual</code>	name of column containing the observed class

Value

A single numeric value.

Author(s)

D.E. Beaudette

References

Brier, Glenn W. 1950. "Verification of Forecasts Expressed in Terms of Probability." Monthly Weather Review 78 (1): 1-3. doi:10.1175/1520-0493(1950)078<0001:VOFEIT>2.0.CO;2.

ca630

Soil Data from the Central Sierra Nevada Region of California

Description

Site and laboratory data from soils sampled in the central Sierra Nevada Region of California.

Usage

data(ca630)

Format

List containing:

\$site : A data frame containing site information.

user_site_id national user site id

mlra the MLRA

county the county

ssa soil survey area

lon longitude, WGS84

lat latitude, WGS84

pedon_key national soil profile id

user_pedon_id local soil profile id

cntrl_depth_to_top control section top depth (cm)

cntrl_depth_to_bot control section bottom depth (cm)

sampled_taxon_name soil series name

\$lab : A data frame containing horizon information.

pedon_key national soil profile id

layer_key national horizon id

layer_sequence horizon sequence number

hzn_top horizon top (cm)

hzn_bot horizon bottom (cm)

hzn_desgn horizon name

texture_description USDA soil texture

nh4_sum_bases sum of bases extracted by ammonium acetate (pH 7)

ex_acid exchangeable acidity [method ?]
 CEC8.2 cation exchange capacity by sum of cations method (pH 8.2)
 CEC7 cation exchange capacity by ammonium acetate (pH 7)
 bs_8.2 base saturation by sum of cations method (pH 8.2)
 bs_7 base saturation by ammonium acetate (pH 7)

Details

These data were extracted from the NSSL database. ‘ca630’ is a list composed of site and lab data, each stored as dataframes. These data are modeled by a 1:many (site:lab) relation, with the ‘pedon_id’ acting as the primary key in the ‘site’ table and as the foreign key in the ‘lab’ table.

Note

These data are out of date. Pending some new data + documentation. Use with caution

Source

<https://ncsslalabdatamart.sc.egov.usda.gov/>

Examples

```
## Not run:
library(plyr)
library(lattice)
library(Hmisc)
library(maps)
library(sp)

# check the data out:
data(ca630)
str(ca630)

# note that pedon_key is the link between the two tables

# make a copy of the horizon data
ca <- ca630$lab

# promote to a SoilProfileCollection class object
depths(ca) <- pedon_key ~ hzn_top + hzn_bot

# add site data, based on pedon_key
site(ca) <- ca630$site

# ID data missing coordinates: '|' is a logical OR
(missing.coords.idx <- which(is.na(ca$lat) | is.na(ca$lon)))

# remove missing coordinates by safely subsetting
if(length(missing.coords.idx) > 0)
ca <- ca[-missing.coords.idx, ]
```

```

# register spatial data
coordinates(ca) <- ~ lon + lat

# assign a coordinate reference system
proj4string(ca) <- '+proj=longlat +datum=NAD83'

# check the result
print(ca)

# map the data (several ways to do this, here is a simple way)
map(database='county', region='california')
points(coordinates(ca), col='red', cex=0.5)

# aggregate %BS 7 for all profiles into 1 cm slices
a <- slab(ca, fm= ~ bs_7)

# plot median & IQR by 1 cm slice
xyplot(
  top ~ p.q50, data=a, lower=a$p.q25, upper=a$p.q75,
  ylim=c(160,-5), alpha=0.5, scales=list(alternating=1, y=list(tick.num=7)),
  panel=panel.depth_function, prepanel=prepanel.depth_function,
  ylab='Depth (cm)', xlab='Base Saturation at pH 7',
  par.settings=list(superpose.line=list(col='black', lwd=2))
)

# aggregate %BS at pH 8.2 for all profiles by MLRA, along 1 cm slices
# note that mlra is stored in @site
a <- slab(ca, mlra ~ bs_8.2)

# keep only MLRA 18 and 22
a <- subset(a, subset=mlra %in% c('18', '22'))

# plot median & IQR by 1 cm slice, using different colors for each MLRA
xyplot(
  top ~ p.q50, groups=mlra, data=a, lower=a$p.q25, upper=a$p.q75,
  ylim=c(160,-5), alpha=0.5, scales=list(y=list(tick.num=7, alternating=3), x=list(alternating=1)),
  panel=panel.depth_function, prepanel=prepanel.depth_function,
  ylab='Depth (cm)', xlab='Base Saturation at pH 8.2',
  par.settings=list(superpose.line=list(col=c('black','blue'), lty=c(1,2), lwd=2)),
  auto.key=list(columns=2, title='MLRA', points=FALSE, lines=TRUE)
)

# safely compute hz-thickness weighted mean CEC (pH 7)
# using data.frame objects
head(lab.agg.cec_7 <- ddply(ca630$lab, .(pedon_key),
  .fun=summarise, CEC_7=wtd.mean(bs_7, weights=hzn_bot-hzn_top)))

# extract a SPDF with horizon data along a slice at 25 cm
s.25 <- slice(ca, fm=25 ~ bs_7 + CEC7 + ex_acid)
splot(s.25, zcol=c('bs_7','CEC7','ex_acid'))

# note that the ordering is preserved:

```

```
all.equal(s.25$pedon_key, profile_id(ca))

# extract a data.frame with horizon data at 10, 20, and 50 cm
s.multiple <- slice(ca, fm=c(10,20,50) ~ bs_7 + CEC7 + ex_acid)

# Extract the 2nd horizon from all profiles as SPDF
ca.2 <- ca[, 2]

# subset profiles 1 through 10
ca.1.to.10 <- ca[1:10, ]

# basic plot method: profile plot
plot(ca.1.to.10, name='hzn_desgn')

## End(Not run)
```

checkSPC

Test for a valid SoilProfileCollection

Description

Test for a valid SoilProfileCollection

Usage

```
checkSPC(x)
```

Arguments

x a SoilProfileCollection object

Details

Test for valid SoilProfileCollection by checking for slots defined in the class prototype. Likely only used between major versions of ‘aqp’ where internal structure of SoilProfileCollection has changed.

Value

TRUE or FALSE. Consider using rebuildSPC() if FALSE.

Author(s)

D.E. Beaudette

See Also

[rebuildSPC](#)

clod.hz.ids *Return the unique horizon IDs within a "depth-clod"*

Description

clod.hz.ids returns a vector of unique indices corresponding to a depth interval. As arguments, it takes a single-profile SoilProfileCollection 'p', a top depth 'z1' and an optional bottom depth 'z2'.

If just top depth is specified, the unique index of the horizon intersected by that depth is returned. If bottom depth is specified, all horizon IDs that are intersected by the depth interval are returned.

Usage

```
clod.hz.ids(p, z1, z2 = NA, as.list = FALSE)
```

Arguments

p	A single-profile SoilProfileCollection
z1	Top depth (required) - depth to intersect horizon at; if 'z2' specified, top depth of intersect interval.
z2	OPTIONAL: Bottom depth - bottom depth of intersection interval
as.list	OPTIONAL: return a list? default: FALSE

Details

This function is a workhorse used by glom() to create an index of unique horizon IDs intersecting the attribute of interest (depth) at the specified levels. This function returns the horizon IDs required to index an SPC and produce either a subset SPC or data.frame representation of the "clod" returned.

Value

A vector of unique horizon IDs (often numeric or character).

Author(s)

Andrew Gene Brown

See Also

glom, slice, slab

Examples

```

data(sp1, package = 'aqp')
depths(sp1) <- id ~ top + bottom
site(sp1) <- ~ group

p <- sp1[1]

foo <- clod.hz.ids(p, 25, 100)

foo

```

colorQuantiles *Soil Color Range via Quantiles*

Description

Estimate central tendency and spread of soil color using marginal quantiles and L1 median of CIELAB coordinates.

Usage

```

colorQuantiles(soilColors, p = c(0.05, 0.5, 0.95))
plotColorQuantiles(res, title='', mar=c(2,1.5,1,1))

```

Arguments

soilColors	vector of R colors (sRGB colorspace)
p	marginal quantiles of interest
res	list returned by colorQuantiles
title	optional title printed above color quantile figure
mar	customized margins, see details

Details

Colors are converted from sRGB to CIELAB (D65 illuminant), marginal quantiles of L,A,B coordinates are estimated, and L1 median L,A,B is estimates. The closest Munsell chips (via Munsell/CIELAB lookup table provided by `munse11`) and R colors are determined by locating chips closest to the marginal quantiles and L1 median.

The results can be conveniently inspected using `plotColorQuantiles`.

Value

A List containing the following elements:

marginal	data.frame containing marginal quantiles in CIELAB (D65), closest colors and Munsell chips
L1	L1 median CIELAB (D65) values, closest color and Munsell chip

Note

This is still rather experimental.

Author(s)

D.E. Beaudette

Examples

```
## Not run:
# example data, see manual page for details
data(sp5)

# slice top 25 cm
s <- slice(sp5, 1:25 ~ .)

# check some of the data
par(mar=c(0,0,0,0))
plot(sample(s, 25), divide.hz=FALSE, name='', print.id=FALSE, width=0.5)

# colors
previewColors(unique(s$soil_color))

# compute marginal quantiles and L1 median
cq <- colorQuantiles(s$soil_color)

# simple graphical display of results
plotColorQuantiles(cq)

## End(Not run)
```

confusionIndex

Confusion Index

Description

Calculate the confusion index of Burrough et al., 1997.

Usage

```
confusionIndex(x)
```

Arguments

x vector of probabilities [0,1], should not contain NA

Value

A single numeric value.

Author(s)

D.E. Beaudette

References

Burrough, P.A., P.F.M. van Gaans, and R. Hootsmans. 1997. "Continuous Classification in Soil Survey: Spatial Correlation, Confusion and Boundaries." *Geoderma* 77: 115-35. doi:10.1016/S0016-7061(97)00018-9.

Examples

```
# a very simple example
p <- c(0.25, 0.25, 0.4, 0.05, 0.05)

confusionIndex(p)
```

crit.clay.argillic *Determines threshold (minimum) clay content for argillic upper bound*

Description

Given a vector or matrix of "eluvial" horizon clay contents (%), `crit.clay.argillic()` returns a vector or matrix of minimum clay contents (thresholds) that must be met for an argillic horizon clay increase.

Uses the standard equations for clay contents less than 15 %, between 15 and 40 %, and greater than 40 %. Based on the clay increase criteria in the definition of the argillic horizon from 12th Edition Keys to Soil Taxonomy (Soil Survey Staff, 2014).

Usage

```
crit.clay.argillic(eluvial_clay_content)
```

Arguments

eluvial_clay_content

A numeric vector or matrix containing clay contents of potential "eluvial" horizons. May contain NA.

Value

A vector or matrix (input-dependent) containing minimum "illuvial" horizon clay contents (thresholds) to be met for argillic horizon clay increase.

Note

This function is intended for identifying clay content threshold required for an argillic horizon. These thresholds may not apply depending on the specifics of your soil. E.g. if the upper part of argillic has been plowed (has Ap immediately over upper boundary) the clay increase requirement can be waived (Soil Survey Staff, 2014).

Author(s)

Andrew Gene Brown

References

Soil Survey Staff. 2014. Keys to Soil Taxonomy, 12th ed. USDA-Natural Resources Conservation Service, Washington, DC.

See Also

[getArgillicBounds](#), [get.increase.matrix](#)

Examples

```
# crit.clay.argillic uses different equations for clay content
# less than 15 %, between 15 and 40 %, and >40 %

crit.clay.argillic(eluvial_clay_content=c(5, 20, 45))
```

estimatePSCS

Estimate boundaries of the particle size control section (U.S Soil Taxonomy; 12th edition)

Description

Estimates the upper and lower boundary of the particle size control section by applying a programmatic version of the particle size control section key from the Keys to Soil Taxonomy (12th edition).

Requires information to identify argillic horizons (clay contents, horizon designations) with `getArgillicBounds()` as well as the presence of plow layers and surface organic soil material. Any `getArgillicBounds()` arguments may be passed to `estimatePSCS`.

Requires information on taxonomic order (to handle andisols).

WARNING: Soils in arenic or grossarenic subgroups, with fragipans, or with strongly contrasting PSCs may not be classified correctly. The author would welcome a dataset to develop this functionality for.

Usage

```
estimatePSCS(p, attr = "clay", hzdesgn = "hzname", tax_order_field = "tax_order",
bottom.pattern = "Cr|R|Cd", ...)
```

Arguments

<code>p</code>	A single-profile <code>SoilProfileCollection</code> object
<code>attr</code>	Name of the horizon attribute containing clay contents. Default 'clay'
<code>hzdesgn</code>	Name of the horizon attribute containing the horizon designation. Default 'hz-name'
<code>tax_order_field</code>	Name of the site attribute containing taxonomic order; for handling PSCS rules for Andisols in lieu of lab data. May be NA or column missing altogether, in which case Andisol PSC possibility is ignored.
<code>bottom.pattern</code>	Regular expression pattern to match a root-restrictive contact. Default matches Cr, R or Cd. This argument is passed to both <code>estimateSoilDepth</code> and <code>getArgillicBounds</code> .
<code>...</code>	additional arguments are passed to <code>getArgillicBounds()</code>

Value

A numeric vector containing the top and bottom depth of the particle size control section. First value is top, second value is bottom.

Author(s)

Andrew Gene Brown

References

Soil Survey Staff. 2014. Keys to Soil Taxonomy, 12th ed. USDA-Natural Resources Conservation Service, Washington, DC.

See Also

`getArgillicBounds`, `getSurfaceHorizonDepth`

Examples

```
data(sp1, package = 'aqp')
depths(sp1) <- id ~ top + bottom
site(sp1) <- ~ group

p <- sp1[1]
attr <- 'prop' # clay contents
foo <- estimatePSCS(p, hzdesgn='name', attr = attr)
foo
```

estimateSoilDepth *Estimate Soil Depth*

Description

Estimate the soil depth of a single profile within a SoilProfileCollection object. This function would typically be called by [profileApply](#).

Usage

```
estimateSoilDepth(f, name = "hzname", top = "hzdept", bottom = "hzdepb", p = 'Cr|R|Cd',
  no.contact.depth = NULL, no.contact.assigned = NULL)
```

Arguments

f	A SoilProfileCollection object of length 1, e.g. a single profile
name	the name of the column that contains horizon designations
top	the name of the column that contains horizon top depths
bottom	the name of the column that contains horizon bottom depths
p	a REGEX pattern for determining contact with bedrock
no.contact.depth	in the absense of contact with bedrock, a depth at which we can assume a standard depth
no.contact.assigned	assumed standard depth

Value

a single integer, the soil depth

Author(s)

D.E. Beaudette and J.M. Skovlin

See Also

[getSoilDepthClass](#), [profileApply](#)

Examples

```
data(sp1)
depths(sp1) <- id ~ top + bottom

# apply to each profile in a collection, and save as site-level attribute
sp1$depth <- profileApply(sp1, estimateSoilDepth, name='name', top='top', bottom='bottom')

# this function can be used to "find" depth to any feature
```

```

# that can be defined via REGEX pattern matching on the horizon name
# for example, locate the depth to the top "Bt" horizon
# returning NA when there is no match
sp1$top_Bt <- profileApply(sp1, estimateSoilDepth, name='name', top='top',
                          bottom='bottom', p='Bt', no.contact.depth=0, no.contact.assigned=NA)

# reduced margins
par(mar=c(1,1,1,2))
# adjust default y-offset and depth scaling for following examples
plot(sp1, y.offset=10, scaling.factor=0.5)

# get plotting parameters for profile widths and depth scaling factors
lsp <- get("last_spc_plot", envir = aqp.env)

# positions on x-axis, same for both depth and top "Bt" horizon
x.positions <- (1:length(sp1)) - lsp$width

# annotate contact with unicode right-arrow
# y-position is adjusted based on plot y-offset and scaling factor
y.positions <- lsp$y.offset + (sp1$depth * lsp$scaling.factor)
text(x.positions, y.positions, '\u2192', col='red', adj=1, cex=1.25, lwd=2)

# annotate top "Bt" depth with unicode right-arrow
# y-position is adjusted based on plot y-offset and scaling factor
y.positions <- lsp$y.offset + (sp1$top_Bt * lsp$scaling.factor)
text(x.positions, y.positions, '\u2192', col='blue', adj=1, cex=1.25, lwd=2)

## Not run:
# sample data
data(gopheridge, package='soilDB')

# run on a single profile
estimateSoilDepth(gopheridge[1, ])

# apply to an entire collection
profileApply(gopheridge, estimateSoilDepth)

## End(Not run)

```

evalGenHZ

Evaluate Generalized Horizon Labels

Description

Data-driven evaluation of generalized horizon labels using nMDS and silhouette width.

Usage

```

evalGenHZ(obj, genhz, vars, non.matching.code = "not-used",
stand = TRUE, trace = FALSE, metric = "euclidean")

```

Arguments

<code>obj</code>	a <code>SoilProfileCollection</code> object
<code>genhz</code>	name of horizon-level attribute containing generalized horizon labels
<code>vars</code>	character vector of horizon-level attributes to include in the evaluation
<code>non.matching.code</code>	code used to represent horizons not assigned a generalized horizon label
<code>stand</code>	standardize variables before computing distance matrix (default = TRUE), passed to daisy
<code>trace</code>	verbose output from passed to isoMDS , (default = FALSE)
<code>metric</code>	distance metric, passed to daisy

Details

Non-metric multidimensional scaling is performed via [isoMDS](#). The input distance matrix is generated by [daisy](#) using (complete cases of) horizon-level attributes from `obj` as named in `vars`.

Silhouette widths are computed via [silhouette](#). The input distance matrix is generated by [daisy](#) using (complete cases of) horizon-level attributes from `obj` as named in `vars`. Note that observations with `genhz` labels specified in `non.matching.code` are removed filtered before calculation of the distance matrix.

Value

a list is returned containing:

horizons `c('mds.1', 'mds.2', 'sil.width', 'neighbor')`

stats mean and standard deviation of `vars`, computed by generalized horizon label

dist the distance matrix as passed to [isoMDS](#)

Author(s)

D.E. Beaudette

See Also

[get.ml.hz](#)

evalMissingData	<i>Evaluate Missing Data</i>
-----------------	------------------------------

Description

Evaluate missing data in a SoilProfileCollection object

Usage

```
evalMissingData(x, vars, name = "hzname", p = "Cr|R|Cd")
```

Arguments

x	a SoilProfileCollection object
vars	a character vector naming horizon-level attributes in x
name	the name of a horizon-level attribute where horizon designations are stored
p	REGEX pattern used to match non-soil horizons

Details

Data completeness is evaluated by profile, based on the thickness of horizons with complete horizon-level attribute values (specified in vars) divided by the total thickness. The default REGEX pattern, p, should catch most non-soil horizons which are excluded from the evaluation. Elements of vars that do not exist in horizonNames(x) are ignored.

Value

A vector values ranging from 0 to 1, representing the percentage of non-NA data (as specified in vars) for each profile.

Author(s)

D.E. Beaudette

Examples

```
# example data
data(sp2)
# init SPC object
depths(sp2) <- id ~ top + bottom
# compute data completeness
sp2$data.complete <- evalMissingData(sp2, vars = c('r', 'g', 'b'), name = 'name')
# rank
new.order <- order(sp2$data.complete)
# plot along data completeness ranking
plot(sp2, plot.order=new.order)
# add axis, note re-ordering of axis labels
axis(side=1, at=1:length(sp2), labels = round(sp2$data.complete[new.order], 2),
line=-2, cex.axis=0.75)
```

f.noise

*Example Objective Function for Full-Pattern Matching***Description**

Basic objective function that can be used as a starting point for developing XRD full-pattern matching strategies. [details pending...]

Usage

```
f.noise(inits, pure.patterns, sample.pattern, eps.total = 0.05)
```

Arguments

<code>inits</code>	vector of initial guesses for mineral fractions, last item is a noise component
<code>pure.patterns</code>	a matrix of XRD patterns of pure samples, resampled to the same twotheta resolution and rescaled according to an external standard
<code>sample.pattern</code>	the unknown or composite pattern, aligned to the same twotheta axis as the pure patterns and rescaled to an external standard
<code>eps.total</code>	precision of comparisons; currently not used

Details

This is similar to the work of Chipera and Bish (2002), using the methods described in (Bish, 1994). If the flexibility of a custom objective function is not required, the linear model framework should be sufficient for pattern fitting. GLS should be used if realistic standard errors are needed.

Value

the sum of absolute differences between the unknown pattern and combination of pure patterns for the current set of mixture proportions

Author(s)

Dylan E. Beaudette

References

Chipera, S.J., & Bish, D.L. (2002) FULLPAT: A full-pattern quantitative analysis program for X-ray powder diffraction using measured and calculated patterns. *J. Applied Crystallography*, 35, 744-749.

Bish, D. 1994. Quantitative Methods in Soil Mineralogy, in *Quantitative X-Ray Diffraction Analysis of Soil*. Amonette, J. & Zelazny, L. (ed.) Soil Science Society of America, pp 267-295.

See Also

[resample.twotheta](#)

Examples

```

# sample data
data(rruff.sample)

# get number of measurements
n <- nrow(rruff.sample)

# number of components
n.components <- 6

# mineral fractions, normally we don't know these
w <- c(0.346, 0.232, 0.153, 0.096, 0.049, 0.065)

# make synthetic combined pattern
# scale the pure substances by the known proportions
rruff.sample$synthetic_pat <- apply(sweep(rruff.sample[,2:7], 2, w, '*'), 1, sum)

# add 1 more substance that will be unknown to the fitting process
rruff.sample$synthetic_pat <- rruff.sample$synthetic_pat +
(1 - sum(w)) * rruff.sample[,8]

# try adding some nasty noise
# rruff.sample$synthetic_pat <- apply(sweep(rruff.sample[,2:7], 2, w, '*'), 1, sum) +
# runif(n, min=0, max=100)

# look at components and combined pattern
par(mfcol=c(7,1), mar=c(0,0,0,0))
plot(1:n, rruff.sample$synthetic_pat, type='l', axes=FALSE)
legend('topright', bty='n', legend='combined pattern', cex=2)
for(i in 2:7)
{
plot(1:n, rruff.sample[, i], type='l', axes=FALSE)
legend('topright', bty='n',
legend=paste(names(rruff.sample)[i], '(', w[i-1], ')', sep=''), cex=2)
}

## fit pattern mixtures with a linear model
l <- lm(synthetic_pat ~ nontronite + montmorillonite + clinocllore
+ antigorite + chamosite + hematite, data=rruff.sample)

summary(l)

par(mfcol=c(2,1), mar=c(0,3,0,0))
plot(1:n, rruff.sample$synthetic_pat, type='l', lwd=2, lty=2, axes=FALSE,
xlab='', ylab='')
lines(1:n, predict(l), col=2)
axis(2, cex.axis=0.75, las=2)

```

```

legend('topright', legend=c('original','fitted'), col=c(1,2), lty=c(2,1),
      lwd=c(2,1), bty='n', cex=1.25)

plot(1:n, resid(1), type='l', axes=FALSE, xlab='', ylab='', col='blue')
abline(h=0, col=grey(0.5), lty=2)
axis(2, cex.axis=0.75, las=2)
legend('topright', legend=c('residuals'), bty='n', cex=1.25)

## fitting by minimizing an objective function (not run)

# SANN is a slower algorithm, sometimes gives strange results
# default Nelder-Mead is most robust
# CG is fastest --> 2.5 minutes max
# component proportions (fractions), and noise component (intensity units)
# initial guesses may affect the stability / time of the fit

## this takes a while to run
## synthetic pattern
# o <- optim(par=c(0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1), f.noise,
# method='CG', pure.patterns=rruff.sample[,2:7],
# sample.pattern=rruff.sample$synthetic_pat)
#
#
# # estimated mixture proportions
# o$par
#
# # compare with starting proportions
# rbind(o$par[1:n.components], w)
#
# # if we had an unknown pattern we were trying to match, compare fitted here
# # compute R value 0.1 - 0.2 considered good
# # sum(D^2) / sum(s)
# # o$value / sum(rruff.sample$sample)
#
# # plot estimated mixture vs sample
# # combine pure substances
# pure.mixture <- apply(sweep(rruff.sample[, 2:7], 2, o$par[1:n.components], '*'), 1, sum)
#
# # add in noise
# noise.component <- o$par[n.components+1]
# est.pattern <- pure.mixture + noise.component
#
#
# # plot results
# par(mfcol=c(2,1), mar=c(0,3,0,0))
# plot(1:n, rruff.sample$synthetic_pat, type='l', lwd=2, lty=2, axes=FALSE,
# xlab='', ylab='')
# lines(1:n, est.pattern, col=2)
# lines(1:n, rep(noise.component, n), col=3)

```

```
# axis(2, cex.axis=0.75, las=2)
# legend('topright', legend=c('original', 'fitted', 'noise'), col=c(1,2,3), lty=c(2,1,1),
# lwd=c(2,1,1), bty='n', cex=1.25)
#
# plot(1:n, rruff.sample$synthetic_pat - est.pattern, type='l', axes=FALSE,
# xlab='', ylab='')
# abline(h=0, col=grey(0.5), lty=2)
# axis(2, cex.axis=0.75, las=2)
# legend('topright', legend=c('difference'), bty='n', cex=1.25)
#
```

generalize.hz

Generalize Horizon Names

Description

Generalize a vector of horizon names, based on new classes, and REGEX patterns.

Usage

```
generalize.hz(x, new, pat, non.matching.code, hzdepn)
```

Arguments

x	a character vector of horizon names
new	a character vector of new horizon classes
pat	a character vector of REGEX, same length as x
non.matching.code	text used to describe any horizon not matching any item in pat
hzdepn	a numeric vector of horizon mid-points, must not contain NA, same length as x

Value

factor of the same length as x

Author(s)

Dylan E. Beaudette

References

<http://casoilresource.lawr.ucdavis.edu/>

Examples

```

data(sp1)

# check original distribution of hz designations
table(sp1$name)

# generalize
sp1$genhz <- generalize.hz(sp1$name,
new=c('O', 'A', 'B', 'C', 'R'),
pat=c('O', '^A', '^B', 'C', 'R'))

# see how we did / what we missed
table(sp1$genhz, sp1$name)

```

`get.increase.depths` *Return the horizon top depths from a call to `get.increase.matrix()`*

Description

`get.increase.depths` performs the conversion of the square matrix output of `get.increase.matrix` back to horizon top depth for where criteria were met.

Note that the ‘`threshold.fun`’ result is allowed to contain NA, but that will result in no output for affected cells.

Usage

```
get.increase.depths(p, attr, threshold.fun, vertical.distance)
```

Arguments

<code>p</code>	a <code>SoilProfileCollection</code> , containing a single profile
<code>attr</code>	horizon attribute name to get the "increase" of
<code>threshold.fun</code>	a function that returns the threshold (as a function of <code>attr</code>); may return a constant single value
<code>vertical.distance</code>	the vertical distance (determined from difference SPC top depth variable) within which increase must be met

Value

Returns a numeric vector of depths where the increase requirement is met. Typically the first is the one of interest to taxonomy.

`get.increase.depths` converts to horizon top depth by using above matrix output to determine depths where increase is met.

Author(s)

Andrew Gene Brown

See Also

getArgillicBounds, crit.clay.argillic

Examples

```
data(sp1, package = 'aqp')
depths(sp1) <- id ~ top + bottom
site(sp1) <- ~ group

p <- sp1[[1]]
attr <- 'prop' # clay contents
foo <- get.increase.depths(p, threshold.fun = crit.clay.argillic,
                           attr = attr, vertical.distance = 30)

foo
```

`get.increase.matrix` *compute pair-wise distances to determine where an attribute increases within a vertical distance specified*

Description

Uses matrix outer product to determine all pair-wise differences in 'attr' for the horizons of 'p'. Supplies 'attr' to 'threshold.fun' to determine the minimum value criterion to return TRUE in output matrix for an "increase". Also, computes all pair-wise distances in depth dimension to determine whether the vertical distance criteria have been met simultaneously with 'attr' increase.

This function assumes that the 'threshold.fun' supplied by the user returns either a constant or a vector of equal length to its input.

Note that the 'threshold.fun' result is allowed to contain NA, but that will result in no output for affected cells.

`get.increase.depths` performs the conversion of the square matrix output of `get.increase.matrix` back to horizon top depth for where criteria were met.

Usage

```
get.increase.matrix(p, attr, threshold.fun, vertical.distance)
```

Arguments

<code>p</code>	a SoilProfileCollection, containing a single profile
<code>attr</code>	horizon attribute name to get the "increase" of
<code>threshold.fun</code>	a function that returns the threshold (as a function of attr); may return a constant single value

vertical.distance

the vertical distance (determined from difference SPC top depth variable) within which increase must be met

Value

Returns a square logical matrix reflecting where the increase criteria were met.

get.increase.depths converts to horizon dop depth by using above matrix output to determine depths where increase is met.

Author(s)

Andrew Gene Brown

See Also

getArgillicBounds, crit.clay.argillic

Examples

```
data(sp1, package = 'aqp')
depths(sp1) <- id ~ top + bottom
site(sp1) <- ~ group

p <- sp1[[1]]
attr <- 'prop' # clay contents
foo <- get.increase.matrix(p, threshold.fun = crit.clay.argillic,
                           attr = attr, vertical.distance = 30)

foo
```

get.ml.hz

Determine ML Horizon Boundaries

Description

This function accepts input from slab() along with a vector of horizon names, and returns a data.frame of the most likely horizon boundaries.

Usage

```
get.ml.hz(x, o.names)
```

Arguments

x	output from <code>slab</code>
o.names	an optional character vector of horizon designations that will be used in the final table

Details

This function expects that `x` is a data.frame generated by `slab`. If `x` was not generated by `slab`, then `o.names` is required.

Value

A dataframe with the following columns:

hz	horizon names
top	top boundary
bottom	bottom boundary
confidence	integrated probability over thickness of each ML horizon, rounded to the nearest integer
pseudo.brier	A "pseudo" Brier Score for a multi-class prediction, where the most-likely horizon label is treated as the "correct" outcome. Details on the calculation for traditional Brier Scores here: http://en.wikipedia.org/wiki/Brier_score#Original_definition_by_Brier . Lower values suggest better agreement between ML horizon label and class-wise probabilities.
mean.H	mean Shannon entropy (bits), derived from probabilities within each most-likely horizon. Larger values suggest more confusion within each ML.

Author(s)

D.E. Beaudette

See Also

[slab](#)

Examples

```
data(sp1)
depths(sp1) <- id ~ top + bottom

# normalize horizon names: result is a factor
sp1$name <- generalize.hz(sp1$name,
  new=c('0','A','B','C'),
  pat=c('0', '^A', '^B', 'C'))

# compute slice-wise probability so that it sums to contributing fraction, from 0-150
a <- slab(sp1, fm= ~ name, cpm=1, slab.structure=0:150)

# generate table of ML horizonation
get.ml.hz(a)
```

getArgillicBounds *getArgillicBounds*

Description

getArgillicBounds() estimates upper and lower boundary of argillic diagnostic subsurface horizon for a profile in a single-profile SoilProfileCollection object ('p'). The upper boundary is where the clay increase threshold is met. Uses crit.clay.argillic() as the threshold function for determining whether a clay increase occurs and get.increase.matrix to determine whether the increase is met, whether vertical distance of increase is sufficiently short, and in which horizon.

The lower boundary is first approximated as the depth to a lithic/paralithic/densic contact, or some other horizon matchable by a custom regular expression pattern. Subsequently, that boundary is extended upwards to the end of "evidence of illuviation."

The depth to contact is estimated using 'bottom.pattern' "Cr|R|Cd" by default. It matches anything containing Cr, R or Cd.

The lower gradational horizon regular expression 'lower.grad.pattern' default is '"^[2-9]*CB*[^rt]*\$". It matches anything that starts with a lithologic discontinuity (or none) and a C master horizon designation. May contain B as second horizon designation in transitional horizon. May not contain 'r' or 't' subscript.

There also is an option 'require_t' to omit the requirement for evidence of eluviation in form of 't' subscript in 'hzdesgn'.

Even if "t" subscript is not required for positive identification, the presence of lower gradational C horizons lacking 't' will still be used to modify the lower boundary upward from a detected contact, if needed. If this behavior is not desired, just set 'lower.grad.pattern' to something that will not match any horizons in your data.

Usage

```
getArgillicBounds(p, hzdesgn = "hzname", attr = "clay", require_t = TRUE,
                 bottom.pattern = "Cr|R|Cd",
                 lower.grad.pattern = "^[2-9]*B*CB*[^rtd]*[1-9]*$",
                 as.list = FALSE)
```

Arguments

p	a single-profile SoilProfileCollection
hzdesgn	the name of the column/attribute containing the horizon designation; default="hzname"
attr	the name of the column/attribute containing the clay content; default="clay"
require_t	require a "t" subscript for positive identification of upper and lower bound of argillic? default:
bottom.pattern	regular expression passed to estimateSoilDepth() to match the lower boundary of the soil. default is "Cr R Cd" which approximately matches paralithic, lithic and densic contacts.

`lower.grad.pattern` this is a pattern for adjusting the bottom depth of the argillic horizon upwards from the bottom depth of the soil. The absence of illuviation is used as a final control on horizon pattern matching.

`as.list` return result as a 'list'?

Value

Returns a numeric vector; first value is top depth, second value is bottom depth. If `as.list` is TRUE, returns a list with top depth named "ubound" and bottom depth named "lbound"

Author(s)

Andrew Gene Brown

See Also

`get.increase.matrix`, `get.argillic.increase.depth`, `crit.clay.argillic`

Examples

```
data(sp1, package = 'aqp')
depths(sp1) <- id ~ top + bottom
site(sp1) <- ~ group

p <- sp1[[1]]
attr <- 'prop' # clay contents
foo <- getArgillicBounds(p, hzdesgn='name', attr = attr)
foo
```

`getSoilDepthClass` *Generate Soil Depth Class Matrix*

Description

Generate a boolean matrix of soil depth classes from a `SoilProfileCollection` object.

Usage

```
getSoilDepthClass(f, depth.classes = c(very.shallow = 25,
shallow = 50, mod.deep = 100, deep = 150, very.deep = 1000), ...)
```

Arguments

`f` a `SoilProfileCollection` object

`depth.classes` a named vector of classes and depth breaks

... arguments passed to [estimateSoilDepth](#)

Value

a data.frame containing soil depth and depth class for each profile, see examples

Author(s)

D.E. Beaudette and J.M. Skovlin

See Also

[estimateSoilDepth](#)

Examples

```
data(sp1)
depths(sp1) <- id ~ top + bottom

# generate depth-class matrix
sdc <- getSoilDepthClass(sp1, name='name', top='top', bottom='bottom')

# inspect
head(sdc)

# join back into sp1 as site-level data
site(sp1) <- sdc

## Not run:
# sample data
data(gopheridge, package='soilDB')

getSoilDepthClass(gopheridge)

## End(Not run)
```

getSurfaceHorizonDepth

Determine thickness of horizons (continuous from surface) matching a pattern

Description

This function is used to find the thickness of arbitrary horizon designations that are continuous from the soil surface (depth = 0).

The horizon designation to match is specified with the regular expression pattern 'pattern'. All horizons matching that pattern, that are continuous from the soil surface, count towards the depth / thickness value that is ultimately returned.

getSurfaceHorizonDepth is used by getPlowLayerDepth for matching Ap horizons; and, it is used by getMineralSoilSurfaceDepth to find the thickness of O horizons in lieu of lab data. These examples are taxonomically relevant for both the argillic horizon and the particle size control section.

Usage

```
getSurfaceHorizonDepth(p, pattern, hzdesgn = "hzname")
```

Arguments

p	A single-profile SoilProfileCollection object.
pattern	Regular expression pattern to match for all horizons to be considered part of the "surface"
hzdesgn	The horizon-level attribute name referring to horizon designation. Default 'hzname'.

Value

Returns a numeric value corresponding to the bottom depth of the last horizon matching 'pattern' that is contiguous with other matching horizons up to the soil surface (depth = 0).

Author(s)

Andrew Gene Brown

See Also

getArgillicBounds, estimatePSCS

Examples

```
library(aqp)
data(sp1, package = 'aqp')
depths(sp1) <- id ~ top + bottom
site(sp1) <- ~ group

p <- sp1[1]
q <- sp1[2]

# look at horizon designations in p and q
p$name
q$name

# thickness of all surface horizons containing A
getSurfaceHorizonDepth(p, pattern = 'A', hzdesgn = 'name')

# thickness of all surface horizons that start with A
getSurfaceHorizonDepth(p, pattern = '^A', hzdesgn = 'name')

# thickness of all surface horizons that start with A, and the A is not followed by B
getSurfaceHorizonDepth(p, pattern = '^A[^B]', hzdesgn = 'name')

# thickness of all surface horizons that start with A
# followed by a number from _2_ to 9 (returns ZERO)
getSurfaceHorizonDepth(p, pattern = '^A[2-9]', hzdesgn = 'name')
```

```
# getPlowLayerDepth matches first two horizons in fake Ap horizon data with "buried Ap"
p$a$horizons <- c("Ap1","Ap2","AB", rep('C', nrow(p) - 4), "Apb")
getPlowLayerDepth(p, hzdesgn = 'aphorizons')

# getMineralSoilSurfaceDepth matches first 3 horizons in fake O horizon data
p$o$horizons <- c("Oi1","Oi2","Oe", rep('C', nrow(p) - 4), "2C")
getMineralSoilSurfaceDepth(p, hzdesgn='ohorizons')

# matches first Oi horizon with original horizon designations of pedon 2
getMineralSoilSurfaceDepth(q, hzdesgn='name')
```

glom	<i>Return a "ragged" group of horizons selected from SoilProfileCollection</i>
------	--

Description

glom() returns a "clod" of horizons from a (often single profile) SoilProfileCollection that have a common attribute. You "glom" SPC horizons into a "clod".

A clod is a "ragged" group of horizons. In this case, "ragged" means that number of horizons, horizon depths, distinctness and topography vary from pedon to pedon.

All horizons included within the specified interval are returned in their entirety (not just the portion within the interval). A 'clod' references the original pedon data – it is not resampled like slice or slab.

Horizon intersection is based on unique ID hzidname(spc) and attribute of interest.

Currently, just depth/depth intervals are supported as common attribute for creating a clod i.e. all horizons overlapping interval 25 to 100 cm – but many other attributes are possible and will be implemented perhaps via formula interface similar to slice.

Usage

```
glom(p, z1, z2 = NA, as.data.frame = FALSE)
```

Arguments

p	A single-profile SoilProfileCollection; e.g. glom is called via profileApply()
z1	Top depth (required) - depth to intersect horizon at; if 'z2' specified, top depth of intersect interval.
z2	OPTIONAL: Bottom depth - bottom depth of intersection interval
as.data.frame	OPTIONAL: return a data.frame, by intersection with horizons(p)? default: FALSE

Value

glom() returns a "clod" of horizons that have a common attribute of type data.frame or SoilProfileCollection.

Author(s)

Andrew Gene Brown

See Also[slice](#), [slab](#)**Examples**

```

data(sp1, package = 'aqp')
depths(sp1) <- id ~ top + bottom
site(sp1) <- ~ group

p <- sp1[[1]]

foo <- glom(p, 25, 100)

# there are 4 horizons in the clod glommed from depths 25 to 100 on profile 1 in sp1
nrow(foo)

```

groupedProfilePlot *Grouped Soil Profile Plot*

Description

Plot a collection of soil profiles, sorted and labeled by group.

Usage

```

groupedProfilePlot(x, groups, group.name.offset = -5,
  group.name.cex = 0.75, group.line.col = "RoyalBlue",
  group.line.lwd = 2, group.line.lty = 2, break.style='line',
  arrow.offset=group.name.offset + 5, arrow.length=0.1, ...)

```

Arguments

x	a SoilProfileCollection object
groups	the name of a site-level attribute that defines groups, factor levels will influence plotting order
group.name.offset	vertical offset for group names, single numeric value or vector of offsets
group.name.cex	font size for group names
group.line.col	color for line that splits groups
group.line.lwd	width of line that splits groups
group.line.lty	style of line that splits groups
break.style	style of group boundaries: "line", "arrow", "both"

arrow.offset vertical offset for "arrow" style boundaries, single numeric value or vector of offsets

arrow.length value passed to arrows to define arrow head size

... further arguments to plotSPC

Details

The ordering of groups can be adjusted by converting the site-level attribute used for grouping into a factor and explicitly setting the levels.

Author(s)

D.E. Beaudette

See Also

[plotSPC](#)

Examples

```
# sample data
data(sp1)
# convert colors from Munsell to hex-encoded RGB
sp1$soil_color <- with(sp1, munsell2rgb(hue, value, chroma))

# promote to SoilProfileCollection
depths(sp1) <- id ~ top + bottom
site(sp1) <- ~ group

# plot profiles, sorted and annotated by 'group'
par(mar=c(1,1,1,1))
groupedProfilePlot(sp1, groups='group', max.depth=150, group.name.offset = -12, id.style='side')

# make fake site-level attribute and adjust levels
sp1$new.group <- sample(letters[1:3], size=length(sp1), replace=TRUE)

# tabulate pedons / group
tab <- table(sp1$new.group)

# sort large -> small
tab <- sort(tab, decreasing = TRUE)

# set levels based on sorted tabulation
# assign custom labels
sp1$new.group <- factor(sp1$new.group, levels=names(tab),
labels=paste0(names(tab), '(', tab, ')'))
groupedProfilePlot(sp1, groups='new.group', max.depth=150,
group.name.offset = -10, id.style='side')

# offsets can be set using a vector of values, recycled as needed
groupedProfilePlot(sp1, groups='new.group', max.depth=150,
```

```

group.name.offset=c(-10, -5), id.style='side')

# annotate with arrows instead of vertical lines
groupedProfilePlot(sp1, groups='new.group', max.depth=150,
group.name.offset=c(-10, -12), break.style='arrow', arrow.offset=-3,
group.line.lty = 1, group.line.lwd = 1, id.style='side')

## Not run:
# more complete example using data from soilDB package
data(loafercreek, package='soilDB')
par(mar=c(1,1,1,1))
# lines
groupedProfilePlot(loafercreek, groups='hillslope_pos', group.name.cex = 0.5,
group.name.offset = -10)

# arrows
groupedProfilePlot(loafercreek, groups='hillslope_pos', group.name.cex = 0.5,
group.name.offset = -10, break.style = 'arrow', group.line.lty = 1,
group.line.lwd = 1)

# both
groupedProfilePlot(loafercreek, groups='hillslope_pos', group.name.cex = 0.5,
group.name.offset = -10, break.style = 'both', group.line.lty = 1,
group.line.lwd = 1)

## End(Not run)

```

guessGenHzLevels

Guess Appropriate Ordering for Generalized Horizon Labels

Description

This function makes an (educated) guess at an appropriate set of levels for generalized horizon labels using the median of horizon depth mid-points.

Usage

```
guessGenHzLevels(x, hz = "genhz")
```

Arguments

x	a SoilProfileCollection object
hz	name of horizon-level attribute containing generalized horizon labels, see details

Details

This function is useful when groups of horizons have been generalized via some method other than `generalize.hz`. For example, it may be useful to generalize horizons using labels derived from slice depths. The default sorting of these labels will not follow a logical depth-wise sorting when converted to a factor. `guessGenHzLevels` does a good job of "guessing" the proper ordering of these labels based on median horizon depth mid-point.

Value

a list:

levels a vector of levels sorted by median horizon depth mid-point

median.depths a vector of median horizon mid-points

Author(s)

D.E. Beaudette

See Also

[generalize.hz](#)

Examples

```
# load some example data
data(sp1, package='aqp')

# upgrade to SoilProfileCollection
depths(sp1) <- id ~ top + bottom

# generalize horizon names
n <- c('O', 'A', 'B', 'C')
p <- c('O', 'A', 'B', 'C')
sp1$genhz <- generalize.hz(sp1$name, n, p)

# note: levels are in the order in which originally defined:
levels(sp1$genhz)

# generalize horizons by depth slice
s <- slice(sp1, c(5, 10, 15, 25, 50, 100, 150) ~ .)
s$slice <- paste0(s$top, ' cm')
# not a factor
levels(s$slice)

# the proper ordering of these new labels can be guessed from horizon depths
guessGenHzLevels(s, 'slice')

# convert to factor, and set proper order
s$slice <- factor(s$slice, levels=guessGenHzLevels(s, 'slice')$levels)

# that is better
levels(s$slice)
```

`hzDistinctnessCodeToOffset`*Convert Horizon Distinctness Codes*

Description

This function accepts a vector of horizon distinctness codes, a look-up vector of codes, and a corresponding vector of vertical offsets. The defaults are based on USDA-NCSS field methods.

Usage

```
hzDistinctnessCodeToOffset(x, codes = c("A", "C", "G", "D"), offset = c(0.5, 1.5, 5, 10))
```

Arguments

<code>x</code>	vector of distinctness codes to be converted to offsets
<code>codes</code>	vector of unique distinctness codes
<code>offset</code>	vector of offsets associated with distinctness codes

Details

Missing data (NA) or codes that are not defined in the 'codes' argument are returned as 0 offsets.

Value

a vector of vertical offsets, with the same length as the number of distinctness codes passed to the function

Author(s)

D.E. Beaudette

References

http://www.nrcs.usda.gov/wps/portal/nrcs/detail/soils/ref/?cid=nrcs142p2_054184

See Also

[plotSPC](#)

Examples

```
data(sp1)
hzDistinctnessCodeToOffset(sp1$bound_distinct)
```

 hzTransitionProbabilities

Horizon Transition Probabilities

Description

Functions for creating and working with horizon (sequence) transition probability matrices.

Usage

```
hzTransitionProbabilities(x, name, loopTerminalStates = FALSE)
```

```
genHzTableToAdjMat(tab)
```

```
mostLikelyHzSequence(mc, t0, maxIterations=10)
```

Arguments

x	A SoilProfileCollection object.
name	A horizon level attribute in x that names horizons.
loopTerminalStates	should terminal states loop back to themselves? This is useful when the transition probability matrix will be used to initialize a markovchain object. See examples below.
tab	A cross-tabulation relating original horizon designations to new, generalized horizon designations.
mc	A markovchain object, initialized from a horizon sequence transition probability matrix with looped terminal states.
t0	Time-zero: a label describing an initial state within a markovchain object.
maxIterations	the maximum number of iterations when search for the most-likely horizon sequence

Details

Details and related tutorial pending...

Value

The function `hzTransitionProbabilities` returns a square matrix of transition probabilities. See examples.

The function `genHzTableToAdjMat` returns a square adjacency matrix. See examples.

The function `mostLikelyHzSequence` returns the most likely sequence of horizons, given a markovchain object initialized from horizon transition probabilities and an initial state, `t0`. See examples.

Note

These functions are still experimental and subject to change.

Author(s)

D.E. Beaudette

See Also

[generalize.hz](#)

Examples

```
data(sp4)
depths(sp4) <- id ~ top + bottom

# horizon transition probabilities: row -> col transitions
(tp <- hzTransitionProbabilities(sp4, 'name'))

## Not run:
## plot TP matrix with functions from sharpshootR package
library(sharpshootR)
par(mar=c(0,0,0,0), mfcol=c(1,2))
plot(sp4)
plotSoilRelationGraph(tp, graph.mode = 'directed', edge.arrow.size=0.5)

## demonstrate genhzTableToAdjMat usage
data(loafercreek, package='soilDB')

# convert contingency table -> adj matrix / TP matrix
tab <- table(loafercreek$hzname, loafercreek$genhz)
m <- genhzTableToAdjMat(tab)

# plot
par(mar=c(0,0,0,0), mfcol=c(1,1))
plotSoilRelationGraph(m, graph.mode = 'directed', edge.arrow.size=0.5)

## demonstrate markovchain integration
library(markovchain)
tp.loops <- hzTransitionProbabilities(sp4, 'name', loopTerminalStates = TRUE)

# init new markovchain from TP matrix
mc <- new("markovchain", states=dimnames(tp.loops)[[1]], transitionMatrix = tp.loops)

# simple plot
plot(mc, edge.arrow.size=0.5)

# check absorbing states
absorbingStates(mc)
```

```
# steady-state:
steadyStates(mc)

## End(Not run)
```

missingDataGrid	<i>Missing Data Grid</i>
-----------------	--------------------------

Description

Generate a levelplot of missing data from a SoilProfileCollection object.

Usage

```
missingDataGrid(s, max_depth, vars, filter.column = NULL,
  filter.regex = NULL, cols = NULL, ...)
```

Arguments

s	a SoilProfilecollection object
max_depth	integer specifying the max depth of analysis
vars	character vector of column names over which to evaluate missing data
filter.column	a character string naming the column to apply the filter REGEX to
filter.regex	a character string with a regular expression used to filter horizon data OUT of the analysis
cols	a vector of colors
...	additional arguments passed on to levelplot

Details

This function evaluates a ‘missing data fraction‘ based on slice-wise evaluation of named variables in a SoilProfileCollection object.

Value

A data.frame describing the percentage of missing data by variable.

Note

A lattice graphic is printed to the active output device.

Author(s)

D.E. Beaudette

See Also[slice](#)**Examples**

```
## visualizing missing data
# 10 random profiles
require(plyr)
s <- ldply(1:10, random_profile)

# randomly sprinkle some missing data
s[sample(nrow(s), 5), 'p1'] <- NA
s[sample(nrow(s), 5), 'p2'] <- NA
s[sample(nrow(s), 5), 'p3'] <- NA

# set all p4 and p5 attributes of `soil 1' to NA
s[which(s$id == '1'), 'p5'] <- NA
s[which(s$id == '1'), 'p4'] <- NA

# upgrade to SPC
depths(s) <- id ~ top + bottom

# plot missing data via slicing + levelplot
missingDataGrid(s, max_depth=100, vars=c('p1', 'p2', 'p3', 'p4', 'p5'),
main='Missing Data Fraction')
```

munsell

Munsell to sRGB Lookup Table for Common Soil Colors

Description

A lookup table of interpolated Munsell color chips for common soil colors.

Usage

```
data(munsell)
```

Format

A data frame with 8825 rows.

hue Munsell Hue, upper case

value Munsell Value

chroma Munsell Chroma

r sRGB "red" value (0-1)

g sRGB "green" value (0-1)

b sRGB "blue" value (0-1)

L CIE LAB "L" coordinate
 A CIE LAB "A" coordinate
 B CIE LAB "B" coordinate

Details

See `munsell2rgb` for conversion examples. Note that this table does not currently have entries for values of 2.5—common in most soil color books. These chips should be added in the next major release of `aqp`. Values are referenced to the D65 standard illuminant.

Source

Color chip XYZ values: http://www.rit.edu/cos/colorscience/rc_munsell_renotation.php

References

<http://www.brucelindbloom.com/index.html?ColorCalcHelp.html> Color conversion equations
<http://dx.doi.org/10.1016/j.cageo.2012.10.020> Methods used to generate this table

Examples

```
data(munsell)
```

munsell2rgb

Convert Munsell Notation to and from sRGB color coordinates

Description

Color conversion based on a look-up table of common soil colors.

Usage

```
munsell2rgb(the_hue, the_value, the_chroma, alpha=1,
maxColorValue=1, return_triplets=FALSE)
rgb2munsell(color, colorSpace='LAB', nClosest=1)
parseMunsell(munsellColor, convertColors=TRUE, ...)
getClosestMunsellChip(munsellColor, convertColors=TRUE, ...)
```

Arguments

<code>the_hue</code>	a vector of one or more more hues, upper-case
<code>the_value</code>	a vector of one or more values
<code>the_chroma</code>	a vector of one or more chromas, may be NA for neutral hues
<code>alpha</code>	alpha channel value (for transparency effects)
<code>maxColorValue</code>	maximum sRGB color value (see rgb)

return_triplets	should the function return raw sRGB triplets instead of an R color
color	a data.frame or matrix object containing sRGB coordinates in the range of [0,1]
colorSpace	the colorspace to use for finding the closest chip, LAB is the most accurate, sRGB is another possibility
nClosest	number of closest Munsell colors to return
munsellColor	character vector of strings containing Munsell colors, e.g. '10YR 4/3'
convertColors	logical, should parsed Munsell colors be converted into sRGB values
...	further arguments to munsell2rgb

Details

These functions generalize to vectorized usage, as long as the length of each argument is the same. Both functions will pad output with NA if there are any NA present in the inputs.

Neutral hues are approximated by greyscale shades ranging from 20% (darker) to 80% (lighter). No chroma is required for neutral hues.

Gley soil colors that are missing a chroma will not be correctly interpreted. Consider using a chroma of 1.

Values of "2.5" (common in soil color descriptions) are silently truncated to "2".

Non-standard Munsell colors (e.g. '7.9YR 2.7/2.0') can be matched (nearest-neighbor, no interpolation) to the closest color within the 'munsell' sRGB look-up table via `getClosestMunsellChip()`. See examples below.

Value

For Munsell to sRGB conversion, a vector of R colors is returned that is the same length as the input data. If `return_triplets` is TRUE, then a dataframe (of sample length as input) of r,g,b values is returned.

For sRGB to Munsell conversion, a dataframe (NA-padded) of hue, value, chroma, and Euclidean distance to nearest matching color is returned.

Note

Care should be taken when using the resulting sRGB values; they are close to their Munsell counterparts, but will vary based on your monitor and ambient lighting conditions. Also, the value used for `maxColorValue` will affect the brightness of the colors. The default value (1) will usually give acceptable results, but can be adjusted to force the colors closer to what the user thinks they should look like.

Author(s)

D.E. Beaudette

References

<http://ncss-tech.github.io/AQP/> <http://www.brucelindbloom.com/index.html?ColorCalcHelp.html> <http://www.cis.rit.edu/mcsl/online/munsell.php> <http://www.munsellcolourscienceforpainters.com/MunsellAndKubelkaMunkToolbox/MunsellAndKubelkaMunkToolbox.html>

Examples

```
# Munsell to sRGB triplets:
# function is vectorized as long as arguments are the same length
color <- munsell2rgb(the_hue=c('10YR', '2.5YR'), the_value=c(3, 5),
the_chroma=c(5, 6), return_triplets=TRUE)

# RGB triplets to closest Munsell color (in sRGB space)
# function is vectorized
rgb2munsell(color)

# neutral hues (N) map to approximate greyscale colors
# chroma may be any number or NA
g <- expand.grid(hue='N', value=2:8, chroma=NA, stringsAsFactors=FALSE)
munsell2rgb(g$hue, g$value, g$chroma)

# basic example: no factors!
d <- expand.grid(hue='10YR', value=2:8, chroma=1:8, stringsAsFactors=FALSE)
d$color <- with(d, munsell2rgb(hue, value, chroma))

# similar to the 10YR color book page
plot(value ~ chroma, data=d, col=d$color, pch=15, cex=3)

# multiple pages of hue:
hues <- c('2.5YR', '5YR', '7.5YR', '10YR')
d <- expand.grid(hue=hues, value=2:8, chroma=seq(2,8,by=2), stringsAsFactors=FALSE)
d$color <- with(d, munsell2rgb(hue, value, chroma))

# plot: note that we are setting panel order from red-->yellow
library(lattice)
xyplot(value ~ factor(chroma) | factor(hue, levels=hues),
main="Common Soil Colors", layout=c(4,1), scales=list(alternating=1),
strip=strip.custom(bg=grey(0.85)),
data=d, as.table=TRUE, subscripts=TRUE, xlab='Chroma', ylab='Value',
panel=function(x, y, subscripts, ...)
{
panel.xyplot(x, y, pch=15, cex=4, col=d$color[subscripts])
}
)

# try again, this time annotate with LAB coordinates:
if(require(colorspace)) {
d.rgb <- with(d, munsell2rgb(hue, value, chroma, return_triplets=TRUE))
d.lab <- as(with(d.rgb, sRGB(r,g,b)), 'LAB')
```

```

d <- data.frame(d, d.lab@coords)

xyplot(value ~ factor(chroma) | factor(hue, levels=hues),
main="Common Soil Colors - Annotated with LAB Coordinates", layout=c(4,1),
scales=list(alternating=1), strip=strip.custom(bg=grey(0.85)),
data=d, as.table=TRUE, subscripts=TRUE, xlab='Chroma', ylab='Value',
panel=function(x, y, subscripts, ...) {
  panel.xyplot(x, y, pch=15, cex=7, col=d$color[subscripts])
  lab.text <- with(d[subscripts, ], paste(round(L), round(A), round(B), sep='\n'))
  panel.text(x, y, labels=lab.text, cex=0.75, col='white', font=2)
}
)

# also demonstrate A ~ hue for each slice of chroma
xyplot(A ~ factor(hue, levels=hues) | factor(value), groups=chroma, data=d,
scales=list(alternating=1), strip=strip.custom(bg=grey(0.85)),
main="A-coordinate vs. Munsell Hue", sub='panels are Munsell value, colors are Munsell chroma',
xlab='Munsell Hue', ylab='A-coordinate', pch=16,
type='b', as.table=TRUE, auto.key=list(lines=TRUE, points=FALSE, columns=4))
}

# soils example
data(sp1)

# convert colors
sp1$soil_color <- with(sp1, munsell2rgb(hue, value, chroma))

# simple plot, may need to tweak gamma-correction...
image(matrix(1:nrow(sp1)), axes=FALSE, col=sp1$soil_color, main='Soil Colors')

# convert into a more useful color space
# you will need the colorspace package for this to work
if(require(colorspace)) {
# keep RGB triplets from conversion
sp1.rgb <- with(sp1, munsell2rgb(hue, value, chroma, return_triplets=TRUE))

# convert into LAB color space
sp1.lab <- as(with(sp1.rgb, sRGB(r,g,b)), 'LAB')
plot(sp1.lab)
}

# convert a non-standard color to closest "chip" in `munsell` look-up table
getClosestMunsellChip('7.9YR 2.7/2.0', convertColors = FALSE)
# convert directly to R color
getClosestMunsellChip('7.9YR 2.7/2.0')

```

Description

Panel function for plotting grouped soil property data, along with upper and lower estimates of uncertainty.

Usage

```
panel.depth_function(x, y, id, upper = NA, lower = NA,
  subscripts = NULL, groups = NULL, sync.colors=FALSE, cf=NA,
  cf.col=NA, cf.interval=20, ...)
```

Arguments

x	x values (generated by calling lattice function)
y	y values (generated by calling lattice function)
id	vector of id labels, same length as x and y—only required when plotting segments (see Details section)
upper	vector of upper confidence envelope values
lower	vector of lower confidence envelope values
subscripts	paneling indices (generated by calling lattice function)
groups	grouping data (generated by calling lattice function)
sync.colors	optionally sync the fill color within the region bounded by (lower–upper) with the line colors
cf	optionally annotate contributing fraction data at regular depth intervals see slab
cf.col	optionall color for contributing fraction values, typically used to override the line color
cf.interval	number of depth units to space printed contributing fraction values
...	further arguments to lower-level lattice plotting functions, see below

Details

This function can be used to replace `panel.superpose` when plotting depth function data. When requested, contributing fraction data are printed using colors the same color as corresponding depth function lines unless a single color value is given via `cf.col`.

Author(s)

Dylan E. Beaudette

References

<http://casoilresource.lawr.ucdavis.edu/>

See Also

[sp1](#), [slice](#), [slab](#)

Examples

```

library(lattice)
data(sp1)

# 1. plotting mechanism for step-functions derived from soil profile data
xyplot(cbind(top,bottom) ~ prop, data=sp1,id=sp1$id,
panel=panel.depth_function, ylim=c(250,-10),
scales=list(y=list(tick.number=10)), xlab='Property',
ylab='Depth (cm)', main='panel.depth_function() demo'
)

# 1.1 include groups argument to leverage lattice styling framework
sp1$group <- factor(sp1$group, labels=c('Group 1', 'Group2'))

xyplot(cbind(top,bottom) ~ prop, groups=group, data=sp1, id=sp1$id,
panel=panel.depth_function, ylim=c(250,-10),
scales=list(y=list(tick.number=10)), xlab='Property',
ylab='Depth (cm)', main='panel.depth_function() demo',
auto.key=list(columns=2, points=FALSE, lines=TRUE),
par.settings=list(superpose.line=list(col=c('Orange', 'RoyalBlue'))))
)

```

plotMultipleSPC

Plot Multiple SoilProfileCollection Objects

Description

Combine multiple SoilProfileCollection objects into a single profile sketch, with annotated groups.

Usage

```

plotMultipleSPC(spc.list, group.labels,
args = rep(list(NA), times=length(spc.list)),
arrow.offset = 2, bracket.base.depth = 95,
...)

```

Arguments

spc.list	a list of SoilProfileCollection objects
group.labels	a vector of group labels, one for each SoilProfileCollection object
args	a list of arguments passed to plotSPC, one for each SoilProfileCollection object
arrow.offset	vertical offset in depth from base of start / end profiles and group bracket arrows
bracket.base.depth	baseline depth used for group brackets
...	additional arguments to the first call to plotSPC

Details

See examples below for usage.

Note

Multiple color legends for thematic profile sketches are not currently supported, use with caution.

Author(s)

D.E. Beaudette and Ben Marshall

See Also

[profileGroupLabels](#)

Examples

```
# load sample data
data(sp3)
data(sp4)

# convert soil colors
sp3$h <- NA ; sp3$s <- NA ; sp3$v <- NA
sp3.rgb <- with(sp3, munsell2rgb(hue, value, chroma, return_triplets=TRUE))
sp3[, c('h','s','v')] <- t(with(sp3.rgb, rgb2hsv(r, g, b, maxColorValue=1)))

# promote to SoilProfileCollection
depths(sp3) <- id ~ top + bottom
depths(sp4) <- id ~ top + bottom

# combine into a list
spc.list <- list(sp3, sp4)

# plot multiple SPC objects, with list of named arguments for each call to plotSPC
par(mar=c(1,1,3,3))
plotMultipleSPC(spc.list, group.labels=c('Collection 1', 'Collection 2'),
args=list(list(name='name', id.style='top'),
list(name='name', id.style='side')), bracket.base.depth=120)
```

plot_distance_graph *Between Individual Distance Plot*

Description

Plot pair-wise distances between individuals as line segments.

Usage

```
plot_distance_graph(D, idx = 1:dim(as.matrix((D)))[1])
```

Arguments

D distance matrix, should be of class 'dist' or compatible class
idx an integer sequence defining which individuals should be compared

Details

By default all individuals are plotting on the same axis. When there are more than about 10 individuals, the plot can become quite messy. See examples below for ideas.

Value

No value is returned.

Author(s)

Dylan E Beaudette

References

<http://casoilresource.lawr.ucdavis.edu/>

See Also

[sp2](#), [profile_compare](#)

Examples

```
data(sp2)

d <- profile_compare(sp2, vars=c('prop', 'field_ph', 'hue', 'value'),
max_d=100, k=0.01, sample_interval=5)

par(mfcol=c(3,1), mar=c(2.5,4.5,1,1))
plot_distance_graph(d, idx=1:6)
plot_distance_graph(d, idx=7:12)
plot_distance_graph(d, idx=12:18)
```

previewColors

Preview Colors

Description

Preview colors as a grid arranged according to CIE2000 distances.

Usage

```
previewColors(cols, method='grid', col.order=NULL,  
nrow=ceiling(sqrt(length(cols))), ncol=nrow, border.col='black', pt.cex=2)
```

Arguments

cols	vector of R colors
method	either "grid", "MDS", or "manual", see details
col.order	integer vector used to order colors
nrow	number of rows used by "grid" method
ncol	number of columns used by "grid" method
border.col	border color used by "grid" method
pt.cex	point scaling factor used by "MDS" method

Details

Color sorting is based on CIE2000 distances as calculated by `farver::compare_colour()`. The "grid" method arranges colors in a rectangular grid with ordering based on divisive hierarchical clustering of the pair-wise distances. Unique colors are used when `cols` contains more than 5,000 colors.

The "MDS" method arranges unique colors via classical multidimensional scaling (principal coordinates) via `cmdscale()`.

Colors can be manually arranged by supplying a vector of integers to `col.order` and setting `method='manual'`.

Author(s)

D.E. Beaudette

Examples

```
# example data  
data(sp2)  
  
# convert into SoilProfileCollection object  
depths(sp2) <- id ~ top + bottom  
  
previewColors(sp2$soil_color)  
previewColors(sp2$soil_color, method = 'MDS', pt.cex = 3)
```

profileApply-methods *Apply a function to soil profiles within a SoilProfileCollection object.*

Description

Apply a function to soil profiles within a SoilProfileCollection object, each iteration has access to a SoilProfileCollection object.

Usage

```
# method for SoilProfileCollection objects
profileApply(object, FUN, simplify=TRUE, ...)
```

Arguments

object	a SoilProfileCollection
FUN	a function to be applied to each profile within the collection
simplify	logical, should the result be simplified to a vector? see examples
...	further arguments passed to FUN

Value

When simplify is TRUE, a vector of length `nrow(object)` (horizon data) or of length `length(object)` (site data). When simplify is FALSE, a list is returned.

Methods

```
signature(object = "SoilProfileCollection")
```

See Also

[slab](#), [estimateSoilDepth](#)

Examples

```
data(sp1)
depths(sp1) <- id ~ top + bottom

# estimate soil depth using horizon designations
profileApply(sp1, estimateSoilDepth, name='name', top='top', bottom='bottom')

# scale properties within each profile
# scaled = (x - mean(x)) / sd(x)
sp1$d <- profileApply(sp1, FUN=function(x) round(scale(x$prop), 2))
plot(sp1, name='d')

# compute depth-wise differencing by profile
```

```

# note that our function expects that the column 'prop' exists
f <- function(x) { c(x$prop[1], diff(x$prop)) }
sp1$d <- profileApply(sp1, FUN=f)
plot(sp1, name='d')

# compute depth-wise cumulative sum by profile
# note the use of an anonymous function
sp1$d <- profileApply(sp1, FUN=function(x) cumsum(x$prop))
plot(sp1, name='d')

# compute profile-means, and save to @site
# there must be some data in @site for this to work
site(sp1) <- ~ group
sp1$mean_prop <- profileApply(sp1, FUN=function(x) mean(x$prop, na.rm=TRUE))

# re-plot using ranks defined by computed summaries (in @site)
plot(sp1, plot.order=rank(sp1$mean_prop))

## iterate over profiles, subsetting horizon data

# example data
data(sp1)

# promote to SoilProfileCollection
depths(sp1) <- id ~ top + bottom
site(sp1) <- ~ group

# make some fake site data related to a depth of some importance
sp1$ddep <- profileApply(sp1, function(i) {round(rnorm(n=1, mean=mean(i$top)))})

# custom function for subsetting horizon data, by profile
# keep horizons with lower boundary < site-level attribute 'dep'
fun <- function(i) {
  # extract horizons
  h <- horizons(i)
  # make an expression to subset horizons
  exp <- paste('bottom < ', i$dep, sep='')
  # subset horizons, and write-back into current SPC
  horizons(i) <- subset(h, subset=eval(parse(text=exp)))
  # return modified SPC
  return(i)
}

# list of modified SoilProfileCollection objects
l <- profileApply(sp1, fun, simplify=FALSE)

# re-combine list of SoilProfileCollection objects into a single SoilProfileCollection
sp1.sub <- union(l)

# graphically check
par(mfrow=c(2,1), mar=c(0,0,1,0))

```

```

plot(sp1)
points(1:length(sp1), sp1$dep, col='red', pch=7)
plot(sp1.sub)

## Not run:
##
## helper functions: these must be modified to suit your own data
##

# compute the weighted-mean of some property within a given diagnostic horizon
# note that this requires conditional eval of data that may contain NA
# see ?slab for details on the syntax
# note that function expects certain columns within 'x'
f.diag.wt.prop <- function(x, d.hz, prop) {
  # extract diagnostic horizon data
  d <- diagnostic_hz(x)
  # subset to the requested diagnostic hz
  d <- d[d$diag_kind == d.hz, ]
  # if missing return NA
  if(nrow(d) == 0)
    return(NA)

  # extract depths and check for missing
  sv <- c(d$featdept, d$featdepb)
  if(any(is.na(sv)))
    return(NA)

  # create formula from named property
  fm <- as.formula(paste('~', prop))
  # return just the (weighted) mean, accessed from @horizons
  s <- slab(x, fm, slab.structure=sv, slab.fun=mean)$value
  return(s)
}

# conditional eval of thickness of some diagnostic feature or horizon
# will return a vector of length(x), you can save to @site
f.diag.thickness <- function(x, d.hz) {
  # extract diagnostic horizon data
  d <- diagnostic_hz(x)
  # subset to the requested diagnostic hz
  d <- d[d$diag_kind == d.hz, ]
  # if missing return NA
  if(nrow(d) == 0)
    return(NA)

  # compute thickness
  thick <- d$featdepb - d$featdept
  return(thick)
}

```

```

# conditional eval of property within particle size control section
# makes assumptions about the SPC that is passed-in
f.psc.prop <- function(x, prop) {
  # these are accessed from @site
  sv <- c(x$psctopdepth, x$pscbotdepth)
  # test for missing PCS data
  if(any(is.na(sv)))
    return(NA)

  # this should never happen... unless someone made a mistake
  # check to make sure that the lower PSC boundary is shallower than the depth
  if(sv[2] > max(x))
    return(NA)

  # create formula from named property
  fm <- as.formula(paste('~', prop))
  # return just the (weighted) mean, accessed from @horizons
  s <- slab(x, fm, slab.structure=sv, slab.fun=mean)$value
  return(s)
}

# try with some sample data
data(loafercreek, package='soilDB')

profileApply(loafercreek, f.diag.wt.prop, d.hz='argillic horizon', prop='clay')
profileApply(loafercreek, f.diag.thickness, d.hz='argillic horizon')
profileApply(loafercreek, f.psc.prop, prop='clay')

## End(Not run)

```

profileGroupLabels *Soil Profile Group Labels*

Description

Labels groups of soil profiles within soil profile sketches.

Usage

```
profileGroupLabels(x0, x1, labels, y0 = 100,
y1 = 98, label.offset = 2, label.cex = 0.75)
```

Arguments

x0	integer indices to the first profile within each group
x1	integer indices to the last profile within each group
labels	vector of group labels
y0	baseline depth used for group brackets

y1	depth used for start and end markers for group brackets (see examples)
label.offset	vertical offset of group labels from baseline
label.cex	label size

Details

See examples below for ideas.

Note

This function is typically called by some other convenience function such as [plotMultipleSPC](#).

Author(s)

D.E. Beaudette

See Also

[plotMultipleSPC](#)

Examples

```
# load sample data
data(sp3)
data(sp4)

# convert soil colors
sp3$h <- NA ; sp3$s <- NA ; sp3$v <- NA
sp3.rgb <- with(sp3, munsell2rgb(hue, value, chroma, return_triplets=TRUE))
sp3[, c('h','s','v')] <- t(with(sp3.rgb, rgb2hsv(r, g, b, maxColorValue=1)))

# promote to SoilProfileCollection
depths(sp3) <- id ~ top + bottom
depths(sp4) <- id ~ top + bottom

# combine into a list
spc.list <- list(sp3, sp4)

# compute group lengths and start/stop locations
n.groups <- length(spc.list)
spc.lengths <- sapply(spc.list, length)
n.pedons <- sum(spc.lengths)
group.starts <- c(1, 1 + cumsum(spc.lengths[-n.groups]))
group.ends <- cumsum(spc.lengths)

# determine depths of first / last profile in each group
yy <- unlist(sapply(spc.list, function(i) profileApply(i, max)))
tick.heights <- yy[c(group.starts, group.ends)] + 2

# plot 2 SoilProfileCollection objects on the same axis
par(mar=c(1,1,1,1))
```

```

plot(sp3, n=n.pedons)
plot(sp4, add=TRUE, x.idx.offset=group.ends[1], plot.depth.axis=FALSE, id.style='side')
# annotate groups
profileGroupLabels(x0=group.starts, x1=group.ends,
labels=c('Collection 1', 'Collection 2'), y0=120, y1=tick.heights)

```

profile_compare-methods

Numerical Soil Profile Comparison

Description

Performs a numerical comparison of soil profiles using named properties, based on a weighted, summed, depth-segment-aligned dissimilarity calculation. If `s` is a [SoilProfileCollection](#), site-level variables (2 or more) can also be used. The site-level and horizon-level dissimilarity matrices are then re-scaled and averaged.

Usage

```

pc(s, vars, max_d, k, filter=NULL, sample_interval=NA,
replace_na=TRUE, add_soil_flag=TRUE,
return_depth_distances=FALSE, strict_hz_eval=FALSE,
progress='none', plot.depth.matrix=FALSE, rescale.result=FALSE,
verbose=FALSE)

```

Arguments

<code>s</code>	a dataframe with at least 2 columns of soil properties, and an 'id' column for each profile. horizon depths must be integers and self-consistent
<code>vars</code>	A vector with named properties that will be used in the comparison. These are typically column names describing horizon-level attributes (2 or more), but can also contain site-level attributes (2 or more) if <code>s</code> is a SoilProfileCollection .
<code>max_d</code>	depth-slices up to this depth are considered in the comparison
<code>k</code>	a depth-weighting coefficient, use '0' for no depth-weighting (see examples below)
<code>filter</code>	an index used to determine which horizons (rows) are included in the analysis
<code>sample_interval</code>	use every n-th depth slice instead of every depth slice, useful for working with > 1000 profiles at a time
<code>replace_na</code>	if TRUE, missing data are replaced by maximum dissimilarity (TRUE)
<code>add_soil_flag</code>	The algorithm will generate a 'soil'/'non-soil' matrix for use when comparing soil profiles with large differences in depth (TRUE). See details section below.
<code>return_depth_distances</code>	return intermediate, depth-wise dissimilarity results (FALSE)
<code>strict_hz_eval</code>	should horizons be strictly checked for internal self-consistency? (FALSE)

progress	'none' (default): argument passed to <code>ddply</code> and related functions, see create_progress_bar for all possible options; 'text' is usually fine.
plot.depth.matrix	should a plot of the 'soil'/'non-soil' matrix be returned (FALSE)
rescale.result	should the result be rescaled by dividing by <code>max(D)</code> (FALSE)
verbose	extra debug output (FALSE)

Details

Variability in soil depth can interfere significantly with the calculation of between-profile dissimilarity—what is the numerical “distance” (or dissimilarity) between a slice of soil from profile A and the corresponding, but missing, slice from a shallower profile B? Gower’s distance metric would yield a NULL distance, despite the fact that intuition suggests otherwise: shallower soils should be more dissimilar from deeper soils. For example, when a 25 cm deep profile is compared with a 50 cm deep profile, numerical distances are only accumulated for the first 25 cm of soil (distances from 26 - 50 cm are NULL). When summed, the total distance between these profiles will generally be less than the distance between two profiles of equal depth. Our algorithm has an option (setting `replace_na=TRUE`) to replace NULL distances with the maximum distance between any pair of profiles for the current depth slice. In this way, the numerical distance between a slice of soil and a corresponding slice of non-soil reflects the fact that these two materials should be treated very differently (i.e. maximum dissimilarity).

This alternative calculation of dissimilarities between soil and non-soil slices solves the problem of comparing shallow profiles with deeper profiles. However, it can result in a new problem: distances calculated between two shallow profiles will be erroneously inflated beyond the extent of either profile’s depth. Our algorithm has an additional option (setting `add_soil_flag=TRUE`) that will preserve NULL distances between slices when both slices represent non-soil material. With this option enabled, shallow profiles will only accumulate mutual dissimilarity to the depth of the deeper profile.

Note that when the `add_soil_flag` option is enabled (default), slices are classified as 'soil' down to the maximum depth to which at least one of variables used in the dissimilarity calculation is not NA. This will cause problems when profiles within a collection contain all NAs within the columns used to determine dissimilarity. An approach for identifying and removing these kind of profiles is presented in the examples section below.

A notice is issued if there are any NA values within the matrix used for distance calculations, as these values are optionally replaced by the max dissimilarity.

Our approach builds on the work of (Moore, 1972) and the previously mentioned depth-slicing algorithm.

Value

A dissimilarity matrix object of class 'dissimilarity, dist', optionally scaled by `max(D)`.

Methods

- `data = "SoilProfileCollection"` see [SoilProfileCollection](#)
- `data = "data.frame"` see [profile_compare](#)

Author(s)

Dylan E. Beaudette

References

1. <http://casoilresource.lawr.ucdavis.edu/>
2. Moore, A.; Russell, J. & Ward, W. Numerical analysis of soils: A comparison of three soil profile models with field classification. *Journal of Soil Science*, 1972, 23, 194-209.

See Also

[slice](#), [daisy](#)

Examples

```
## 1. check out the influence depth-weight coef:
require(lattice)
z <- rep(1:100,4)
k <- rep(c(0,0.1,0.05,0.01), each=100)
w <- 100*exp(-k*z)

xyplot(z ~ w, groups=k, ylim=c(105,-5), xlim=c(-5,105), type='l',
       ylab='Depth', xlab='Weighting Factor',
       auto.key=list(columns=4, lines=TRUE, points=FALSE, title="k", cex=0.8, size=3),
       panel=function(...) {
         panel.grid(h=-1,v=-1)
         panel.superpose(...)
       }
)

## 2. basic implementation, requires at least two properties
# implementation for a data.frame class object
data(sp1)
d <- profile_compare(sp1, vars=c('prop','group'), max_d=100, k=0.01,
plot.depth.matrix=TRUE)

# upgrade to SoilProfileCollection
depths(sp1) <- id ~ top + bottom
op <- par(mfrow=c(1,2))
# perform comparison on SoilProfileCollection object
# compare soil/non-soil matrix plot
d <- profile_compare(sp1, vars=c('prop','group'), max_d=100, k=0.01,
plot.depth.matrix=TRUE)

# plot profile collection
plot(sp1)
# annotate max depth of profile comparison
abline(h=100, col='red', lty=2)
par(op)
```

```

# more soil properties
data(sp2)
d.1 <- profile_compare(sp2, vars=c('prop', 'field_ph', 'hue', 'value'),
max_d=100, k=0.01, plot.depth.matrix=TRUE)

# add some missing data:
sp2$prop[1:2] <- NA
d.2 <- profile_compare(sp2, vars=c('prop', 'field_ph', 'hue', 'value'),
max_d=100, k=0.01, plot.depth.matrix=TRUE)

# note small changes in D:
cor(d.1, d.2)

## 3. identify profiles within a collection that contain all NAs
require(plyr)
s <- ldply(1:10, random_profile)
depths(s) <- id ~ top + bottom

# replace first profile's data with NA
na.required <- nrow(s[1, ])
s$p1[1:na.required] <- NA
s$p2[1:na.required] <- NA

# attempt profile comparison: this won't work, throws an error
# d <- profile_compare(s, vars=c('p1', 'p2'), max_d=100, k=0)

# check for soils that are missing all clay / total RF data
f.check.NA <- function(i) length(which(is.na(i$p1) | is.na(i$p2))) / nrow(i) == 1
missing.too.much.data.idx <- which(profileApply(s, f.check.NA))

# remove bad profiles and try again: works
s.no.na <- profile_compare(s[-missing.too.much.data.idx, ], vars=c('p1', 'p2'),
max_d=100, k=0, plot.depth.matrix=TRUE)

## 4. better plotting of dendrograms with ape package:
if(require(ape) & require(cluster) & require(MASS)) {
h <- diana(d)
p <- as.phylo(as.hclust(h))
plot(ladderize(p), cex=0.75, label.offset=1)
tiplabels(col=cutree(h, 3), pch=15)

## 5. other uses of the dissimilarity matrix
# Sammon Mapping: doesn't like '0' values in dissimilarity matrix
d.sam <- sammon(d)

# simple plot
dev.off() ; dev.new()
plot(d.sam$points, type = "n", xlim=range(d.sam$points[,1] * 1.5))
text(d.sam$points, labels=row.names(as.data.frame(d.sam$points)),
cex=0.75, col=cutree(h, 3))

}

```

```
## 6. try out the 'sample_interval' argument
# compute using sucessively larger sampling intervals
data(sp3)
d <- profile_compare(sp3, vars=c('clay','cec','ph'),
max_d=100, k=0.01)
d.2 <- profile_compare(sp3, vars=c('clay','cec','ph'),
max_d=100, k=0.01, sample_interval=2)
d.10 <- profile_compare(sp3, vars=c('clay','cec','ph'),
max_d=100, k=0.01, sample_interval=10)
d.20 <- profile_compare(sp3, vars=c('clay','cec','ph'),
max_d=100, k=0.01, sample_interval=20)

# check the results via hclust / dendrograms
oldpar <- par(mfcol=c(1,4), mar=c(2,1,2,2))
plot(as.dendrogram(hclust(d)), horiz=TRUE, main='Every Depth Slice')
plot(as.dendrogram(hclust(d.2)), horiz=TRUE, main='Every 2nd Depth Slice')
plot(as.dendrogram(hclust(d.10)), horiz=TRUE, main='Every 10th Depth Slice')
plot(as.dendrogram(hclust(d.20)), horiz=TRUE, main='Every 20th Depth Slice')
par(oldpar)
```

random_profile

Random Profile

Description

Generate a random soil profile according to set criteria, with correlated depth trends.

Usage

```
random_profile(id, n = c(3, 4, 5, 6), min_thick = 5,
max_thick = 30, n_prop = 5, exact = FALSE, method = 'random_walk',
HzDistinctSim=FALSE, ...)
```

Arguments

id	a character or numeric id used for this profile
n	vector of possible number of horizons, or the exact number of horizons (see below)
min_thick	minimum thickness criteria for a simulated horizon
max_thick	maximum thickness criteria for a simulated horizon
n_prop	number of simulated soil properties (columns in the returned dataframe)
exact	should the exact number of requested horizons be generated? (defaults to FALSE)
method	named method used to synthesize depth function ('random_walk' or 'LPP'), see details
HzDistinctSim	optionally simulate horizon boundary distinctness codes
...	additional parameters passed-in to the LPP (.lpp) function

Details

The random walk method produces profiles with considerable variation between horizons and is based on values from the normal distribution seeded with means and standard deviations drawn from the uniform distribution of [0, 10].

The logistic power peak (LPP) function can be used to generate random soil property depth functions that are sharply peaked. LPP parameters can be hard-coded using the optional arguments: "lpp.a", "lpp.b", "lpp.u", "lpp.d", "lpp.e". Amplitude of the peak is controlled by ("lpp.a + "lpp.b"), depth of the peak by "lpp.u", and abruptness by "lpp.d" and "lpp.e". Further description of the method is outlined in (Brenton et al, 2011). Simulated horizon distinctness codes are based on the USDA-NCSS field description methods (http://www.nrcs.usda.gov/wps/portal/nrcs/detail/?cid=nrcs142p2_054184). Simulated distinctness codes are constrained according to horizon thickness, i.e. a gradual boundary (+/- 5cm) will not be simulated for horizons that are thinner than 3x this vertical distance

Value

A dataframe with the simulated profile.

Note

See examples for ideas on simulating several profiles at once.

Author(s)

Dylan E. Beaudette

References

Myers, D. B.; Kitchen, N. R.; Sudduth, K. A.; Miles, R. J.; Sadler, E. J. & Grunwald, S. Peak functions for modeling high resolution soil profile data Geoderma, 2011, 166, 74-83.

See Also

[profile_compare](#), [hzDistinctnessCodeToOffset](#)

Examples

```
# generate 10 random profiles with default settings:
require(plyr)
d <- ldply(1:10, random_profile)

# add a fake color
d$soil_color <- 'white'

# promote to SoilProfileCollection and plot
depths(d) <- id ~ top + bottom
plot(d)

# simulate horizon boundary distinctness codes:
```

```

d <- ldply(1:10, random_profile, HzDistinctSim=TRUE)
d$soil_color <- grey(0.85)
depths(d) <- id ~ top + bottom
d$HzD <- hzDistinctnessCodeToOffset(d$HzDistinctCode)
plot(d, hz.distinctness.offset='HzD')

# depth functions are generated using the LPP function
opar <- par(mfrow=c(2,1), mar=c(0,0,3,0))
# generate data
d <- ldply(1:10, random_profile, n=c(6, 7, 8), n_prop=1, method='LPP')

# promote to SPC and plot
depths(d) <- id ~ top + bottom
plot(d, color='p1')

# do this again, this time set all of the LPP parameters
d <- ldply(1:10, random_profile, n=c(6, 7, 8), n_prop=1, method='LPP',
lpp.a=5, lpp.b=10, lpp.d=5, lpp.e=5, lpp.u=25)

depths(d) <- id ~ top + bottom
plot(d, color='p1')

# reset plotting defaults
par(opar)

# try plotting the LPP-derived simulated data
# aggregated over all profiles
a <- slab(d, fm= ~ p1)
a$mid <- with(a, (top + bottom) / 2)

library(lattice)
(p1 <- xyplot(mid ~ p.q50, data=a,
lower=a$p.q25, upper=a$p.q75, ylim=c(150,-5), alpha=0.5,
panel=panel.depth_function, prepanel=prepanel.depth_function,
cf=a$contributing_fraction, xlab='Simulated Data', ylab='Depth',
main='LPP(a=5, b=10, d=5, e=5, u=25)',
par.settings=list(superpose.line=list(col='black', lwd=2))
))

# optionally add original data as step-functions
if(require(latticeExtra)) {
  h <- horizons(d)
  p1 + as.layer(xyplot(top ~ p1, groups=id, data=h,
horizontal=TRUE, type='S',
par.settings=list(superpose.line=list(col='blue', lwd=1, lty=2))))
}

```

rebuildSPC	<i>Rebuild a SoilProfileCollection object</i>
------------	---

Description

Rebuild a SoilProfileCollection object

Usage

```
rebuildSPC(x)
```

Arguments

x a SoilProfileCollection object

Details

Attempt rebuilding a SoilProfileCollection object by splitting into components and re-assembling. Likely only used to fix outdated SoilProfileCollection objects that are missing slots.

Value

A valid SoilProfileCollection object.

Author(s)

D.E. Beaudette

See Also

[checkSPC](#)

resample.twotheta	<i>Resample an XRD Pattern</i>
-------------------	--------------------------------

Description

Resample an XRD pattern along a user-defined twotheta resolution via local spline interpolation.

Usage

```
resample.twotheta(twotheta, x, tt.min = min(twotheta),  
tt.max = max(twotheta), new.res = 0.02)
```

Arguments

twotheta	a vector of twotheta value
x	a vector of diffraction intensities corresponding with twotheta values
tt.min	new minimum twotheta value, defaults to current minimum
tt.max	new maximum twotheta value, defaults to current maximum
new.res	new twotheta resolution, defaults to 0.02

Details

Sometimes XRD patterns are collected at different resolutions, or at a resolution that is too great for full pattern matching. This function can be used to resample patterns to a consistent twotheta resolution, or to decimate massive patterns.

Value

A dataframe with the following columns

twotheta	new sequence of twotheta values
x	resampled diffraction intensities

Author(s)

Dylan E Beaudette

References

<http://casoilresource.lawr.ucdavis.edu/>

See Also

[rruff.sample](#)

Examples

```
data(rruff.sample)

# resample single pattern
nontronite.resamp <- with(rruff.sample,
  resample.twotheta(twotheta, nontronite, new.res=0.02) )

# plot original vs. resampled pattern
plot(nontronite ~ twotheta, data=rruff.sample, type='l', col='grey')
lines(nontronite.resamp, col='blue')
```

`rruff.sample`*Sample XRD Patterns*

Description

Several sample XRD patterns from the RRUFF project site.

Usage

```
data(rruff.sample)
```

Format

A data frame with 3000 observations on the following 8 variables.

`twotheta` twotheta values

`nontronite` XRD pattern for nontronite

`montmorillonite` XRD pattern for montmorillonite

`clinochlore` XRD pattern for clinochlore

`antigorite` XRD pattern for antigorite

`chamosite` XRD pattern for chamosite

`hematite` XRD pattern for hematite

`goethite` XRD pattern for goethite

Source

<http://rruff.info/>

References

<http://rruff.info/>

Examples

```
data(rruff.sample)

# plot all patterns
matplot(rruff.sample, type='l', lty=1)
```

shannonEntropy	<i>Shannon Entropy</i>
----------------	------------------------

Description

A very simple implementation of Shannon entropy.

Usage

```
shannonEntropy(x, b = 2)
```

Arguments

x	vector of probabilities [0,1], must sum to 1, should not contain NA
b	logarithm base

Details

0s are automatically removed by `na.rm=TRUE`, as $(0 * \log(0) = \text{Nan})$

Value

A single numeric value.

Note

The result is the normalized Shannon entropy when `b = length(x)` (Kempen et al, 2009).

References

Kempen, Bas, Dick J. Brus, Gerard B.M. Heuvelink, and Jetse J. Stoorvogel. 2009. "Updating the 1:50,000 Dutch Soil Map Using Legacy Soil Data: A Multinomial Logistic Regression Approach." *Geoderma* 151: 311-26. doi:10.1016/j.geoderma.2009.04.023

Shannon, Claude E. (July-October 1948). "A Mathematical Theory of Communication". *Bell System Technical Journal*. 27 (3): 379-423. doi:10.1002/j.1538-7305.1948.tb01338.x

Examples

```
# a very simple example
p <- c(0.25, 0.25, 0.4, 0.05, 0.05)

shannonEntropy(p)
```

sim *Simulate Soil Profiles*

Description

Simulate a collection of soil profiles based on the horization of a single soil profile.

Usage

```
sim(x, n=1, iterations=25, hz.sd=2, min.thick=2)
```

Arguments

x	a SoilProfileCollection object containing a single profile from which to draw simulated data
n	the number of requested simulations
iterations	sampling iterations used to determine each horizon thickness
hz.sd	standard deviation used to simulate horizon thickness, can be a vector but must divide evenly into the number of horizons found in x
min.thick	minumum horizon thickness allowed in simulation results

Details

This function generates a collection of simulated soil profiles based on the horizon thickness data associated with a single "template" profile. Simulation is based on sampling from a family of gaussian distribution with means defined by the "template" profile and standard deviation defined by the user.

Value

A SoilProfileCollection object with n simulated profiles, each containing the same number of horizons and same data as x.

Author(s)

D. E. Beaudette

See Also

[random_profile](#)

Examples

```

# load sample data and convert into SoilProfileCollection
data(sp3)
depths(sp3) <- id ~ top + bottom

# select a profile to use as the basis for simulation
s <- sp3[3, ]

# reset horizon names
s$name <- paste('H', seq_along(s$name), sep='')

# simulate 25 new profiles, using 's' and function defaults
sim.1 <- sim(s, n=25)

# simulate 25 new profiles using 's' and variable SD for each horizon
sim.2 <- sim(s, n=25, hz.sd=c(1, 2, 5, 5, 5, 10, 2))

# plot
par(mfrow=c(2,1), mar=c(0, 0, 0, 0))
plot(sim.1)
mtext('SD = 2', side=2, line=-1.5, font=2, cex=0.75)
plot(sim.2)
mtext('SD = c(1, 2, 5, 5, 5, 10, 2)', side=2, line=-1.5, font=2, cex=0.75)

# aggregate horization of simulated data
# note: set class_prob_mode=2 as profiles were not defined to a constant depth
sim.2$name <- factor(sim.2$name)
a <- slab(sim.2, ~ name, class_prob_mode=2)

# convert to long format for plotting simplicity
library(reshape)
a.long <- melt(a, id.vars=c('top','bottom'), measure.vars=levels(sim.2$name))

# plot horizon probabilities derived from simulated data
# dashed lines are the original horizon boundaries
library(lattice)
xyplot(top ~ value, groups=variable, data=a.long, subset=value > 0,
ylim=c(100, -5), type=c('l','g'), asp=1.5,
ylab='Depth (cm)', xlab='Probability',
auto.key=list(columns=4, lines=TRUE, points=FALSE),
panel=function(...) {
  panel.xyplot(...)
  panel.abline(h=s$stop, lty=2, lwd=2)
})

```

Description

Aggregate soil properties along user-defined ‘slabs’, and optionally within groups.

Usage

```
# method for SoilProfileCollection objects
slab(object, fm, slab.structure=1, strict=FALSE,
      slab.fun=.slab.fun.numeric.default, cpm=1, weights=NULL, ...)
```

Arguments

<code>object</code>	a <code>SoilProfileCollection</code>
<code>fm</code>	A formula: either ‘groups ~ var1 + var2 + var3’ where named variables are aggregated within ‘groups’ OR where named variables are aggregated across the entire collection ‘~ var1 + var2 + var3’. If ‘groups’ is a factor it must not contain NA.
<code>slab.structure</code>	A user-defined slab thickness (defined by an integer), or user-defined structure (numeric vector). See details below.
<code>strict</code>	logical: should horizons be strictly checked for self-consistency?
<code>slab.fun</code>	Function used to process each ‘slab’ of data, ideally returning a vector with names attribute. Defaults to a wrapper function around <code>hdquantile</code> . See details.
<code>cpm</code>	Strategy for normalizing slice-wise probabilities, dividing by either: number of profiles with data at the current slice (<code>cpm=1</code>), or by the number of profiles in the collection (<code>cpm=2</code>). Mode 1 values will always sum to the contributing fraction, while mode 2 values will always sum to 1.
<code>weights</code>	Column name containing weights. NOT YET IMPLEMENTED
<code>...</code>	further arguments passed to <code>slab.fun</code>

Details

Multiple continuous variables OR a single categorical (factor) variable can be aggregated within a call to `slab`. Basic error checking is performed to make sure that top and bottom horizon boundaries make sense. User-defined aggregate functions (`slab.fun`) should return a named vector of results. A new, named column will appear in the results of `slab` for every named element of a vector returned by `slab.fun`. See examples below for a simple example of a slab function that computes mean, mean-1SD and mean+1SD. The default slab function wraps `hdquantile` from the `Hmisc` package, which requires at least 2 observations per chunk. Note that if ‘group’ is a factor it must not contain NAs.

Sometimes `slab` is used to conveniently re-arrange data vs. aggregate. This is performed by specifying `identity` in `slab.fun`. See examples below for a demonstration of this functionality.

Execution time scales linearly (slower) with the total number of profiles in `object`, and exponentially (faster) as the number of profiles / group is increased. `slab()` and `slice()` are much faster and require less memory if input data are either numeric or character.

There are several possible ways to define slabs, using `slab.structure`:

a single integer e.g. 10: data are aggregated over a regular sequence of 10-unit thickness slabs

a vector of 2 integers e.g. `c(50, 60)`: data are aggregated over depths spanning 50–60 units

a vector of 3 or more integers e.g. `c(0, 5, 10, 50, 100)`: data are aggregated over the depths spanning 0–5, 5–10, 10–50, 50–100 units

Value

Output is returned in long format, such that slice-wise aggregates are returned once for each combination of grouping level (optional), variable described in the `fm` argument, and depth-wise 'slab'.

Aggregation of numeric variables, using the default slab function:

variable The names of variables included in the call to `slab`.

groupname The name of the grouping variable when provided, otherwise a fake grouping variable named 'all.profiles'.

p.q5 The slice-wise 5th percentile.

p.q25 The slice-wise 25th percentile

p.q50 The slice-wise 50th percentile (median)

p.q75 The slice-wise 75th percentile

p.q95 The slice-wise 95th percentile

top The slab top boundary.

bottom The slab bottom boundary.

contributing_fraction The fraction of profiles contributing to the aggregate value, ranges from $1/n_{\text{profiles}}$ to 1.

When a single factor variable is used, slice-wise probabilities for each level of that factor are returned as:

variable The names of variables included in the call to `slab`.

groupname The name of the grouping variable when provided, otherwise a fake grouping variable named 'all.profiles'.

A The slice-wise probability of level A

B The slice-wise probability of level B

...

n The slice-wise probability of level n

top The slab top boundary.

bottom The slab bottom boundary.

contributing_fraction The fraction of profiles contributing to the aggregate value, ranges from $1/n_{\text{profiles}}$ to 1.

Methods

data = "SoilProfileCollection" Typical usage, where input is a [SoilProfileCollection](#).

Note

Arguments to `slab` have changed with `aqp` 1.5 (2012-12-29) as part of a code clean-up and optimization. Calculation of weighted-summaries was broken in `aqp` 1.2-6 (2012-06-26), and removed as of `aqp` 1.5 (2012-12-29). `slab` replaced the previously defined `soil.slot.multiple` function as of `aqp` 0.98-8.58 (2011-12-21).

Author(s)

D.E. Beaudette

References

D.E. Beaudette, P. Roudier, A.T. O'Geen, Algorithms for quantitative pedology: A toolkit for soil scientists, *Computers & Geosciences*, Volume 52, March 2013, Pages 258-268, 10.1016/j.cageo.2012.10.020.
Harrell FE, Davis CE (1982): A new distribution-free quantile estimator. *Biometrika* 69:635-640.

See Also

[slice](#), [hdquantile](#)

Examples

```
##
## basic examples
##
library(lattice)
library(grid)

# load sample data, upgrade to SoilProfileCollection
data(sp1)
depths(sp1) <- id ~ top + bottom

# aggregate entire collection with two different segment sizes
a <- slab(sp1, fm = ~ prop)
b <- slab(sp1, fm = ~ prop, slab.structure=5)

# check output
str(a)

# stack into long format
ab <- make.groups(a, b)
ab$which <- factor(ab$which, levels=c('a','b'),
labels=c('1-cm Interval', '5-cm Interval'))

# plot median and IQR
# custom plotting function for uncertainty viz.
xyplot(top ~ p.q50 | which, data=ab, ylab='Depth',
xlab='median bounded by 25th and 75th percentiles',
lower=ab$p.q25, upper=ab$p.q75, ylim=c(250,-5),
panel=panel.depth_function,
prepanel=prepanel.depth_function,
```

```

cf=ab$contributing_fraction,
layout=c(2,1), scales=list(x=list(alternating=1))
)

###
### re-arrange data / no aggregation
###

# load sample data, upgrade to SoilProfileCollection
data(sp1)
depths(sp1) <- id ~ top + bottom

# arrange data by ID
a <- slab(sp1, fm = id ~ prop, slab.fun=identity)

# convert id to a factor for plotting
a$id <- factor(a$id)

# check output
str(a)

# plot via step function
xyplot(top ~ value | id, data=a, ylab='Depth',
        ylim=c(250, -5), as.table=TRUE,
        panel=panel.depth_function,
        prepanel=prepanel.depth_function,
        scales=list(x=list(alternating=1))
)

##
## categorical variable example
##
library(reshape)

# normalize horizon names: result is a factor
sp1$name <- generalize.hz(sp1$name,
new=c('O','A','B','C'),
pat=c('O', '^A', '^B', 'C'))

# compute slice-wise probability so that it sums to contributing fraction, from 0-150
a <- slab(sp1, fm= ~ name, cpm=1, slab.structure=0:150)

# reshape into long format for plotting
a.long <- melt(a, id.vars=c('top','bottom'), measure.vars=c('O','A','B','C'))

# plot horizon type proportions using panels
xyplot(top ~ value | variable, data=a.long, subset=value > 0,
        ylim=c(150, -5), type=c('S','g'), horizontal=TRUE, layout=c(4,1), col=1 )

# again, this time using groups
xyplot(top ~ value, data=a.long, groups=variable, subset=value > 0,

```

```

ylim=c(150, -5), type=c('S','g'), horizontal=TRUE, asp=2)

# adjust probability to size of collection, from 0-150
a.1 <- slab(sp1, fm= ~ name, cpm=2, slab.structure=0:150)

# reshape into long format for plotting
a.1.long <- melt(a.1, id.vars=c('top','bottom'), measure.vars=c('O','A','B','C'))

# combine aggregation from `cpm` modes 1 and 2
g <- make.groups(cmp.mode.1=a.long, cmp.mode.2=a.1.long)

# plot horizon type proportions
xyplot(top ~ value | variable, groups=which, data=g, subset=value > 0,
ylim=c(240, -5), type=c('S','g'), horizontal=TRUE, layout=c(4,1),
auto.key=list(lines=TRUE, points=FALSE, columns=2),
par.settings=list(superpose.line=list(col=c(1,2))),
scales=list(alternating=3))

# apply slice-wise evaluation of max probability, and assign ML-horizon at each slice
(gen.hz.ml <- get.ml.hz(a, c('O','A','B','C'))))

## Not run:
##
## multivariate examples
##
data(sp3)

# add new grouping factor
sp3$group <- 'group 1'
sp3$group[as.numeric(sp3$id) > 5] <- 'group 2'
sp3$group <- factor(sp3$group)

# upgrade to SPC
depths(sp3) <- id ~ top + bottom
site(sp3) <- ~ group

# custom 'slab' function, returning mean +/- 1SD
mean.and.sd <- function(values) {
m <- mean(values, na.rm=TRUE)
s <- sd(values, na.rm=TRUE)
upper <- m + s
lower <- m - s
res <- c(mean=m, lower=lower, upper=upper)
return(res)
}

# aggregate several variables at once, within 'group'
a <- slab(sp3, fm=group ~ L + A + B, slab.fun=mean.and.sd)

# check the results:
# note that 'group' is the column containing group labels
library(lattice)

```

```

xyplot(
  top ~ mean | variable, data=a, groups=group,
  lower=a$lower, upper=a$upper, sync.colors=TRUE, alpha=0.5,
  cf=a$contributing_fraction,
  ylim=c(125,-5), layout=c(3,1), scales=list(x=list(relation='free')),
  par.settings=list(superpose.line=list(lwd=2, col=c('RoyalBlue', 'Orange2'))),
  panel=panel.depth_function,
  prepanel=prepanel.depth_function,
  auto.key=list(columns=2, lines=TRUE, points=FALSE)
)

# compare a single profile to the group-level aggregate values
a.1 <- slab(sp3[1, ], fm=group ~ L + A + B, slab.fun=mean.and.sd)

# manually update the group column
a.1$group <- 'profile 1'

# combine into a single data.frame:
g <- rbind(a, a.1)

# plot with customized line styles
xyplot(
  top ~ mean | variable, data=g, groups=group, subtitles=TRUE,
  lower=a$lower, upper=a$upper, ylim=c(125,-5),
  layout=c(3,1), scales=list(x=list(relation='free')),
  panel=panel.depth_function,
  prepanel=prepanel.depth_function,
  sync.colors=TRUE, alpha=0.25,
  par.settings=list(superpose.line=list(col=c('orange', 'royalblue', 'black'),
    lwd=2, lty=c(1,1,2))),
  auto.key=list(columns=3, lines=TRUE, points=FALSE)
)

## convert mean value for each variable into long format
library(reshape)

# note that depths are no longer in order
a.wide <- cast(a, group + top + bottom ~ variable, value=c('mean'))

## again, this time for a user-defined slab from 40-60 cm
a <- slab(sp3, fm=group ~ L + A + B, slab.structure=c(40,60), slab.fun=mean.and.sd)

# now we have weighted average properties (within the defined slab)
# for each variable, and each group
(a.wide <- cast(a, group + top + bottom ~ variable, value=c('mean'))))

## this time, compute the weighted mean of selected properties, by profile ID
a <- slab(sp3, fm= id ~ L + A + B, slab.structure=c(40,60), slab.fun=mean.and.sd)
(a.wide <- cast(a, id + top + bottom ~ variable, value=c('mean'))))

```

```

## aggregate the entire collection, using default slab function (hdquantile)
## note the missing left-hand side of the formula
a <- slab(sp3, fm= ~ L + A + B)

## weighted-aggregation -- NOT YET IMPLEMENTED --
# load sample data, upgrade to SoilProfileCollection
data(sp1)
depths(sp1) <- id ~ top + bottom

# generate pretend weights as site-level attribute
set.seed(10101)
sp1$site.wts <- runif(n=length(sp1), min=20, max=100)

## End(Not run)

```

slice-methods

Slicing of SoilProfileCollection Objects

Description

Slicing of SoilProfileCollection Objects

Usage

```

# method for SoilProfileCollection objects
slice(object, fm, top.down=TRUE, just.the.data=FALSE, strict=TRUE)

```

Arguments

object	a SoilProfileCollection
fm	A formula: either ‘integer.vector ~ var1 + var2 + var3’ where named variables are sliced according to ‘integer.vector’ OR where all variables are sliced according to ‘integer.vector’ ‘integer.vector ~.’.
top.down	Logical, should slices be defined from the top-down? The default is usually what you want.
just.the.data	Logical, return just the sliced data or a new SoilProfileCollection object.
strict	Logical, should the horizonation be strictly checked for self-consistency?

Value

Either a new SoilProfileCollection with data sliced according to fm, or a data.frame.

Methods

data = "SoilProfileCollection" Typical usage, where input is a [SoilProfileCollection](#).

Note

slab() and slice() are much faster and require less memory if input data are either numeric or character.

Author(s)

D.E. Beaudette

References

D.E. Beaudette, P. Roudier, A.T. O'Geen, Algorithms for quantitative pedology: A toolkit for soil scientists, Computers & Geosciences, Volume 52, March 2013, Pages 258-268, 10.1016/j.cageo.2012.10.020.

See Also

[slab](#)

Examples

```
# simulate some data, IDs are 1:20
library(plyr)
d <- ldply(1:20, random_profile)

# init SoilProfilecollection object
depths(d) <- id ~ top + bottom
head(horizons(d))

# generate single slice at 10 cm
# output is a SoilProfilecollection object
s <- slice(d, 10 ~ name + p1 + p2 + p3)

# generate single slice at 10 cm, output data.frame
s <- slice(d, 10 ~ name + p1 + p2 + p3, just.the.data=TRUE)

# generate integer slices from 0 - 25 cm
s <- slice(d, 0:25 ~ name + p1 + p2 + p3)
plot(s)

# generate slices from 0 - 10 cm, for all variables
s <- slice(d, 0:10 ~ .)
print(s)

# note that pct missing is computed for each slice,
# if all vars are missing, then NA is returned
d$p1[1:10] <- NA
s <- slice(d, 10 ~ ., just.the.data=TRUE)
print(s)

##
## check sliced data
##
```

```

# test that mean of 1 cm slices property is equal to the
# hz-thickness weighted mean value of that property
data(sp1)
depths(sp1) <- id ~ top + bottom

# get the first profile
sp1.sub <- sp1[which(profile_id(sp1) == 'P009'), ]

# compute hz-thickness wt. mean
hz.wt.mean <- with(horizons(sp1.sub),
sum((bottom - top) * prop) / sum(bottom - top)
)

# hopefully the same value, calculated via slice()
s <- slice(sp1.sub, 0:max(sp1.sub) ~ prop)
hz.slice.mean <- mean(s$prop, na.rm=TRUE)

# same?
if(!all.equal(hz.slice.mean, hz.wt.mean))
stop('there is a bug in slice() !!!')

```

soilColorSignature *Soil Profile Color Signatures*

Description

Generate a color signature for each soil profile in a collection.

Usage

```

soilColorSignature(spc, r = "r", g = "g", b = "b",
method='colorBucket', pam.k=3, RescaleLightnessBy = 1, useProportions=TRUE,
pigmentNames=c('.white.pigment', '.red.pigment', '.green.pigment',
'.yellow.pigment', '.blue.pigment'))

```

Arguments

spc	a SoilProfileCollection object
r	horizon level attribute containing soil color (sRGB) red values
g	horizon level attribute containing soil color (sRGB) green values
b	horizon level attribute containing soil color (sRGB) blue values
method	algorithm used to compute color signature, 'colorBucket', 'depthSlices', or 'pam'
pam.k	number of classes to request from cluster::pam()
RescaleLightnessBy	rescaling factor for CIE LAB L-coordinate
useProportions	use proportions or quantities, see details
pigmentNames	names for resulting pigment proportions or quantities

Details

<http://ncss-tech.github.io/AQP/aqp/soil-color-signatures.html>

Value

For the ‘colorBucket’ method, a `data.frame` object containing:

id column set according to `idname(spc)`

.white.pigment proportion or quantity of CIE LAB L-values

.red.pigment proportion or quantity of CIE LAB positive A-values

.green.pigment proportion or quantity of CIE LAB negative A-values

.yellow.pigment proportion or quantity of CIE LAB positive B-values

.blue.pigment proportion or quantity of CIE LAB negative B-values

Column names can be adjusted with the `pigmentNames` argument.

For the ‘depthSlices’ method ...

For the ‘pam’ method ...

Author(s)

D.E. Beaudette

References

https://en.wikipedia.org/wiki/Lab_color_space

See Also

[munsell2rgb](#)

Examples

```
# trivial example, not very interesting
data(sp1)
depths(sp1) <- id ~ top + bottom

# convert Munsell -> sRGB triplets
rgb.data <- munsell2rgb(sp1$hue, sp1$value, sp1$chroma, return_triplets = TRUE)
sp1$r <- rgb.data$r
sp1$g <- rgb.data$g
sp1$b <- rgb.data$b

# extract color signature
pig <- soilColorSignature(sp1)
```

SoilProfileCollection-class
SoilProfileCollection Class

Description

Basic class for storing soil profile collections, associated site data, and metadata.

Objects from the Class

Objects can be created by calls of the form `new("SoilProfileCollection", ...)`.

Slots

idcol: Object of class "character" the name of the column used to uniquely identify profiles

hzidcol: Object of class "character" the name of the column used to uniquely identify horizons

depthcols: Object of class "character" with the names of columns containing the horizon top and bottom boundaries

metadata: Object of class "data.frame" with collection-level metadata, having a single row, and user-defined columns

horizons: Object of class "data.frame" with 1 or more rows per profile

site: Object of class "data.frame" with 1 row per profile

sp: Object of class "SpatialPoints" with 1 row per profile

diagnostic: Object of class "data.frame" with 0 or more rows per profile

Methods

\$ signature(x = "SoilProfileCollection"): ...

\$<- signature(x = "SoilProfileCollection"): ...

[signature(x = "SoilProfileCollection", i = "ANY", j = "ANY"): ...

[[signature(x = "SoilProfileCollection", i = "ANY", j = "ANY"): ...

[[<- signature(x = "SoilProfileCollection", i = "ANY", j = "ANY"): ...

coordinates<- signature(object = "SoilProfileCollection"): ...

horizonDepths signature(object = "SoilProfileCollection"): ...

horizons signature(object = "SoilProfileCollection"): ...

horizons<- signature(object = "SoilProfileCollection"): ...

idname signature(object = "SoilProfileCollection"): ...

names signature(x = "SoilProfileCollection"): ...

horizonNames signature(object = "SoilProfileCollection"): ...

siteNames signature(object = "SoilProfileCollection"): ...

length signature(x = "SoilProfileCollection"): ...

```

max signature(x = "SoilProfileCollection"): ...
metadata signature(object = "SoilProfileCollection"): ...
metadata<- signature(object = "SoilProfileCollection"): ...
min signature(x = "SoilProfileCollection"): ...
profile_id signature(object = "SoilProfileCollection"): ...
profile_plot signature(object = "SoilProfileCollection"): ...
show signature(object = "SoilProfileCollection"): ...
site signature(object = "SoilProfileCollection"): ...
site<- signature(object = "SoilProfileCollection"): ...
slab signature(data = "SoilProfileCollection"): ...
units signature(object = "SoilProfileCollection"): ...
units<- signature(object = "SoilProfileCollection"): ...

```

Author(s)

Pierre Roudier and Dylan E. Beaudette

Examples

```

# concatenate SoilProfileCollection objects
## Not run:
require(plyr)
d <- ldply(1:10, random_profile)

# promote to SoilProfileCollection and plot
depths(d) <- id ~ top + bottom
plot(d)

# split into new SoilProfileCollection objects by index
d.1 <- d[1, ]
d.2 <- d[2, ]
d.345 <- d[3:5, ]

# recombine, note that profiles are sorted according to ID
d.new <- union(list(d.345, d.1, d.2))
plot(d.new)

## End(Not run)

```

SoilProfileCollection-plotting-methods
Profile Plot

Description

Generate a simple diagram of a soil profile, with annotated horizon names.

Usage

```
plotSPC(x, color='soil_color', width=0.2, name=NULL, label=idname(x),
alt.label=NULL, alt.label.col='black', cex.names=0.5,
cex.depth.axis=cex.names, cex.id=cex.names+(0.2*cex.names),
print.id=TRUE, id.style='auto', plot.order=1:length(x), add=FALSE,
scaling.factor=1, y.offset=0, x.idx.offset=0, n=length(x),
max.depth=ifelse(is.infinite(max(x)), 200, max(x)), n.depth.ticks=5,
shrink=FALSE, shrink.cutoff=3, abbr=FALSE, abbr.cutoff=5, divide.hz=TRUE,
hz.distinctness.offset=NULL, hz.distinctness.offset.col='black',
hz.distinctness.offset.lty=2, axis.line.offset=-2.5,
plot.depth.axis=TRUE, density=NULL, col.label=color,
col.palette = rev(brewer.pal(10, 'Spectral')), col.legend.cex=1,
n.legend=8,
lwd=1, lty=1,
default.color=grey(0.95), ...)
```

Arguments

<code>x</code>	a SoilProfileCollection object
<code>color</code>	the name of the column containing R-compatible color descriptions, or a column containing numeric or categorical data; see details
<code>width</code>	scaling of profile widths
<code>name</code>	the name of the (horizon-level) attribute containing horizon designation labels
<code>label</code>	the name of the (site-level) attribute used to identify profiles in the plot
<code>alt.label</code>	the name of a (site-level) attribute used for secondary annotation
<code>alt.label.col</code>	color used when printing secondary annotation
<code>cex.names</code>	character scaling applied to horizon names
<code>cex.depth.axis</code>	character scaling applied to depth scale
<code>cex.id</code>	character scaling applied to profile id
<code>print.id</code>	should the profile id be printed above each profile? (TRUE)
<code>id.style</code>	profile ID printing style: 'auto' (default) = simple heuristic used to select from: 'top' = centered above each profile, 'side' = 'along the top-left edge of profiles'
<code>plot.order</code>	a vector describing the order in which individual SoilProfile objects from the parent should be plotted

<code>add</code>	add to an existing figure
<code>scaling.factor</code>	vertical scaling of the profile heights
<code>y.offset</code>	vertical offset for top of profiles
<code>x.idx.offset</code>	integer specifying horizontal offset from 0
<code>n</code>	integer describing amount of space along x-axis to allocate, defaults to <code>length(x)</code>
<code>max.depth</code>	suggested lower depth boundary of plot
<code>n.depth.ticks</code>	suggested number of ticks in depth scale
<code>shrink</code>	should long horizon names be shrunk by 80% ?
<code>shrink.cutoff</code>	character length defining long horizon names
<code>abbr</code>	should the profile ID be abbreviated?
<code>abbr.cutoff</code>	suggested minimum length for abbreviated IDs
<code>divide.hz</code>	should horizons be divided with a thin black line? (default is TRUE)
<code>hz.distinctness.offset</code>	column name containing vertical offsets used to depict horizon boundary distinctness (same units as profiles)
<code>hz.distinctness.offset.col</code>	color used to encode horizon distinctness (default is 'black')
<code>hz.distinctness.offset.lty</code>	line style used to encode horizon distinctness (default is 2)
<code>axis.line.offset</code>	horizontal offset applied to depth axis (default is -2.5)
<code>plot.depth.axis</code>	plot depth axis? (default is TRUE)
<code>density</code>	fill density used for horizon color shading, either a single integer or a column name containing integer values (default is NULL, no shading)
<code>col.label</code>	text printed above the color-coded legend
<code>col.palette</code>	color palette used to plot numeric data
<code>col.legend.cex</code>	scaling of color legend
<code>n.legend</code>	approximate number of classes used in the legend
<code>lwd</code>	single numeric value: line width multiplier
<code>lty</code>	single integer: line style
<code>default.color</code>	default horizon fill color used when 'color' attribute is NA
<code>...</code>	other arguments passed into lower level plotting functions

Details

Depth limits (`max.depth`) and number of depth ticks (`n.depth.ticks`) are *suggestions* to the `pretty` function. You may have to tinker with both parameters to get what you want.

The 'side' `id.style` is useful when plotting a large collection of profiles, and/or, when profile IDs are long.

If the column containing horizon designations is not specified (the `name` argument), a column (presumed to contain horizon designation labels) is guessed based on regular expression matching of the pattern `'name'`— this usually works, but it is best to manually specify the name of the column containing horizon designations.

The `color` argument can either name a column containing R-compatible colors, possibly created via `munsell2rgb`, or column containing either numeric or categorical (either factor or character) values. In the second case, values are converted into colors and displayed along with a simple legend above the plot. Note that this functionality makes several assumptions about plot geometry and is most useful in an interactive setting.

The `x.idx.offset` argument can be used to shift a collection of pedons from left to right in the figure. This can be useful when plotting several different `SoilProfileCollection` objects within the same figure. Space must be pre-allocated in the first plotting call, with an offset specified in the second call. See examples below.

Value

A new plot of soil profiles is generated, or optionally added to an existing plot.

Methods

```
signature(x = "SoilProfileCollection")
```

Author(s)

Dylan E. Beaudette

References

<http://casoilresource.lawr.ucdavis.edu/>

See Also

[SoilProfileCollection-class](#), [pretty](#), [hzDistinctnessCodeToOffset](#), [addBracket](#), [profileGroupLabels](#)

Examples

```
data(sp1)

# usually best to adjust margins
par(mar=c(0,0,3,0))

# add color vector
sp1$soil_color <- with(sp1, munsell2rgb(hue, value, chroma))

# promote to SoilProfileCollection
depths(sp1) <- id ~ top + bottom

# plot profiles
plot(sp1, id.style='side')
```

```

# title, note line argument:
title('Sample Data 1', line=1, cex.main=0.75)

# plot profiles without horizon-line divisions
plot(sp1, divide.hz=FALSE)

# add dashed lines illustrating horizon boundary distinctness
sp1$hzD <- hzDistinctnessCodeToOffset(sp1$bound_distinct)
plot(sp1, hz.distinctness.offset='hzD')

# plot horizon color according to some property
data(sp4)
depths(sp4) <- id ~ top + bottom
plot(sp4, color='clay')

# another example
data(sp2)
depths(sp2) <- id ~ top + bottom
site(sp2) <- ~ surface

# label with site-level attribute: `surface`
plot(sp2, label='surface', plot.order=order(sp2$surface))

# example using a categorical attribute
plot(sp2, color = "plasticity")

# plot two SPC objects in the same figure
par(mar=c(1,1,1,1))
# plot the first SPC object and
# allocate space for the second SPC object
plot(sp1, n=length(sp1) + length(sp2))
# plot the second SPC, starting from the first empty space
plot(sp2, x.idx.offset=length(sp1), add=TRUE)

```

soil_minerals

Munsell Colors of Common Soil Minerals

Description

Munsell colors for some common soil minerals.

Usage

```
data("soil_minerals")
```

Format

A data frame with 20 observations on the following 5 variables.

mineral mineral name

color Munsell color
 hue Munsell hue
 value Munsell value
 chroma Munsell chroma

Details

Soil color and other properties including texture, structure, and consistence are used to distinguish and identify soil horizons (layers) and to group soils according to the soil classification system called Soil Taxonomy. Color development and distribution of color within a soil profile are part of weathering. As rocks containing iron or manganese weather, the elements oxidize. Iron forms small crystals with a yellow or red color, organic matter decomposes into black humus, and manganese forms black mineral deposits. These pigments paint the soil (Michigan State Soil). Color is also affected by the environment: aerobic environments produce sweeping vistas of uniform or subtly changing color, and anaerobic (lacking oxygen), wet environments disrupt color flow with complex, often intriguing patterns and points of accent. With depth below the soil surface, colors usually become lighter, yellower, or redder.

Source

http://www.nrcs.usda.gov/wps/portal/nrcs/detail/soils/edu/?cid=nrcs142p2_054286

References

1. Lynn, W.C. and Pearson, M.J., The Color of Soil, The Science Teacher, May 2000. 2. Schwertmann, U. 1993. Relations Between Iron Oxides, Soil Color, and Soil Formation. "Soil Color". SSSA Special Publication no. 31, pages 51–69.

Examples

```
## Not run:
library(aqp)
library(ape)
library(cluster)
library(colorspace)

# load common soil mineral colors
data(soil_minerals)
# convert Munsell to R colors
soil_minerals$col <- munsell2rgb(soil_minerals$hue, soil_minerals$value,
soil_minerals$chroma)

# make a grid for plotting
n <- ceiling(sqrt(nrow(soil_minerals)))
# read from top-left to bottom-right
g <- expand.grid(x=1:n, y=n:1)[1:nrow(soil_minerals),]

# convert Munsell -> sRGB -> LAB
col.rgb <- munsell2rgb(soil_minerals$hue, soil_minerals$value,
soil_minerals$chroma, return_triplets = TRUE)
```

```

col.lab <- as(sRGB(as.matrix(col.rgb)), 'LAB')@coords
row.names(col.lab) <- soil_minerals$mineral

# divisive hierarchical clustering of LAB coordinates
d <- daisy(col.lab)
h <- as.hclust(diana(d))
p <- as.phylo(h)

# plot grid of mineral names / colors
layout(matrix(c(1,2), nrow=1), widths = c(1.25,1))
par(mar=c(1,0,0,1))
plot(g$x, g$y, pch=15, cex=12, axes=FALSE, xlab='', ylab='',
     col=rev(soil_minerals$col[h$order]), xlim=c(0.5,5.5), ylim=c(1.5,5.5))
text(g$x, g$y, rev(soil_minerals$mineral[h$order]), adj=c(0.45,5), cex=1, font=2)
text(g$x, g$y, rev(soil_minerals$color[h$order]), col='white', pos=1, cex=0.85, font=2)
title(main='Common Soil Minerals', line=-2, cex.main=2)
mtext('http://www.nrcs.usda.gov/wps/portal/nrcs/detail/soils/edu/?cid=nrcs142p2_054286',
      side=1, cex=0.75, line=-1.5)
mtext('U. Schwertmann, 1993. SSSA Special Publication no. 31, pages 51--69', side=1,
      cex=0.75, line=-0.5)

# dendrogram + tip labels with mineral colors
plot(p, cex=0.85, label.offset=1, font=1)
tiplabels(pch=15, cex=4, col=soil_minerals$col)

## End(Not run)

```

sp1

Soil Profile Data Example 1

Description

Soil profile data from Pinnacles National Monument, CA.

Usage

```
data(sp1)
```

Format

A data frame with 60 observations on the following 21 variables.

```

group a numeric vector
id a character vector
top a numeric vector
bottom a numeric vector
bound_distinct a character vector

```

bound_topography a character vector
 name a character vector
 texture a character vector
 prop a numeric vector
 structure_grade a character vector
 structure_size a character vector
 structure_type a character vector
 stickiness a character vector
 plasticity a character vector
 field_ph a numeric vector
 hue a character vector
 value a numeric vector
 chroma a numeric vector

References

<http://casoilresource.lawr.ucdavis.edu/>

Examples

```

data(sp1)
# convert colors from Munsell to hex-encoded RGB
sp1$soil_color <- with(sp1, munsell2rgb(hue, value, chroma))

# promote to SoilProfileCollection
depths(sp1) <- id ~ top + bottom
site(sp1) <- ~ group

# re-sample each profile into 1 cm (thick) depth slices
# for the variables 'prop', 'name', 'soil_color'
# result is a SoilProfileCollection object
s <- slice(sp1, 0:25 ~ prop + name + soil_color)

# plot, note slices
plot(s)

# aggregate all profiles along 1 cm depth slices,
# using data from column 'prop'
s1 <- slab(sp1, fm= ~ prop)

# check median & IQR
library(lattice)
xyplot(top ~ p.q50 + p.q25 + p.q75,
data=s1, type='S', horizontal=TRUE, col=1, lty=c(1,2,2),
panel=panel.superpose, ylim=c(110,-5), asp=2)

```

sp2

*Honcut Creek Soil Profile Data***Description**

A collection of 18 soil profiles, consisting of select soil morphologic attributes, associated with a stratigraphic study conducted near Honcut Creek, California.

Usage

```
data(sp2)
```

Format

A data frame with 154 observations on the following 21 variables.

```
id profile id
surface dated surface
top horizon top in cm
bottom horizon bottom in cm
bound_distinct horizon lower boundary distinctness class
bound_topography horizon lower boundary topography class
name horizon name
texture USDA soil texture class
prop field-estimated clay content
structure_grade soil structure grade
structure_size soil structure size
structure_type soil structure type
stickiness stickiness
plasticity plasticity
field_ph field-measured pH
hue Munsell hue
value Munsell value
chroma Munsell chroma
r RGB red component
g RGB green component
b RGB blue component
soil_color R-friendly encoding of soil color
```

Author(s)

Dylan E. Beaudette

Source

Busacca, Alan J.; Singer, Michael J.; Verosub, Kenneth L. 1989. Late Cenozoic stratigraphy of the Feather and Yuba rivers area, California, with a section on soil development in mixed alluvium at Honcut Creek. USGS Bulletin 1590-G.

References

<http://casoilresource.lawr.ucdavis.edu/>

Examples

```
data(sp2)

# convert into SoilProfileCollection object
depths(sp2) <- id ~ top + bottom

# transfer site-level data
site(sp2) <- ~ surface

# generate a new plotting order, based on the dated surface each soil was described on
p.order <- order(sp2$surface)

# plot
par(mar=c(1,0,3,0))
plot(sp2, plot.order=p.order)

# setup multi-figure output
par(mfrow=c(2,1), mar=c(0,0,1,0))

# truncate plot to 200 cm depth
plot(sp2, plot.order=p.order, max.depth=200)
abline(h=200, lty=2, lwd=2)

# compute numerical distances between profiles
# based on select horizon-level properties, to a depth of 200 cm
d <- profile_compare(sp2, vars=c('prop', 'field_ph', 'hue'),
max_d=200, k=0, sample_interval=5, rescale.result=TRUE)

# plot dendrogram with ape package:
if(require(ape) & require(cluster)) {
h <- diana(d)
p <- as.phylo(as.hclust(h))
plot(p, cex=0.75, label.offset=0.01, font=1, direct='down', srt=90, adj=0.5, y.lim=c(-0.125, 0.5))

# add in the dated surface type via color
tiplabels(col=as.numeric(sp2$surface), pch=15)

# based on distance matrix values, YMMV
legend('topleft', legend=levels(sp2$surface), col=1:6, pch=15, bty='n', bg='white', cex=0.75)
}
```

sp3

Soil Profile Data Example 3

Description

Soil samples from 10 soil profiles, taken from the Sierra Foothill Region of California.

Usage

```
data(sp3)
```

Format

A data frame with 46 observations on the following 15 variables.

id soil id
top horizon upper boundary (cm)
bottom horizon lower boundary (cm)
clay clay content
cec CEC by amonium acetate at pH 7
ph pH in 1:1 water-soil mixture
tc total carbon percent
hue Munsell hue (dry)
value Munsell value (dry)
chroma Munsell chroma (dry)
mid horizon midpoint (cm)
ln_tc natural log of total carbon percent
L color: l-coordinate, CIE-LAB colorspace (dry)
A color: a-coordinate, CIE-LAB colorspace (dry)
B color: b-coordinate, CIE-LAB colorspace (dry)
name horizon name
soil_color horizon color

Details

These data were collected to support research funded by the Kearney Foundation of Soil Science.

References

<http://casoilresource.lawr.ucdavis.edu/>

Examples

```

## this example investigates the concept of a "median profile"

# required packages
if(require(ape) & require(cluster)) {

data(sp3)

# generate a RGB version of soil colors
# and convert to HSV for aggregation
sp3$h <- NA ; sp3$s <- NA ; sp3$v <- NA
sp3.rgb <- with(sp3, munsell2rgb(hue, value, chroma, return_triplets=TRUE))
sp3[, c('h','s','v')] <- t(with(sp3.rgb, rgb2hsv(r, g, b, maxColorValue=1)))

# promote to SoilProfileCollection
depths(sp3) <- id ~ top + bottom

# aggregate across entire collection
a <- slab(sp3, fm= ~ clay + cec + ph + h + s + v, slab.structure=10)

# check
str(a)

# convert back to wide format
library(reshape)
a.wide.q25 <- cast(a, top + bottom ~ variable, value=c('p.q25'))
a.wide.q50 <- cast(a, top + bottom ~ variable, value=c('p.q50'))
a.wide.q75 <- cast(a, top + bottom ~ variable, value=c('p.q75'))

# add a new id for the 25th, 50th, and 75th percentile pedons
a.wide.q25$id <- 'Q25'
a.wide.q50$id <- 'Q50'
a.wide.q75$id <- 'Q75'

# combine original data with "mean profile"
vars <- c('top','bottom','id','clay','cec','ph','h','s','v')
# make data.frame version of sp3
sp3.df <- as(sp3, 'data.frame')
sp3.grouped <- rbind(
sp3.df[, vars], a.wide.q25[, vars], a.wide.q50[, vars], a.wide.q75[, vars]
)

# re-constitute the soil color from HSV triplets
# convert HSV back to standard R colors
sp3.grouped$soil_color <- with(sp3.grouped, hsv(h, s, v))

# give each horizon a name
sp3.grouped$name <- paste(round(sp3.grouped$clay), '/',
round(sp3.grouped$cec), '/', round(sp3.grouped$ph,1))

```

```

## perform comparison, and convert to phylo class object
## D is rescaled to [0,]
d <- profile_compare(sp3.grouped, vars=c('clay', 'cec', 'ph'), max_d=100,
k=0.01, replace_na=TRUE, add_soil_flag=TRUE, rescale.result=TRUE)

h <- agnes(d, method='ward')
p <- ladderize(as.phylo(as.hclust(h)))

# look at distance plot-- just the median profile
plot_distance_graph(d, 12)

# similarity relative to median profile (profile #12)
round(1 - (as.matrix(d)[12, ] / max(as.matrix(d)[12, ])), 2)

## make dendrogram + soil profiles
# first promote to SoilProfileCollection
depths(sp3.grouped) <- id ~ top + bottom

# setup plot: note that D has a scale of [0,1]
par(mar=c(1,1,1,1))
p.plot <- plot(p, cex=0.8, label.offset=3, direction='up', y.lim=c(2,0),
x.lim=c(1.25,length(sp3.grouped)+1), show.tip.label=FALSE)

# get the last plot geometry
lastPP <- get("last_plot.phylo", envir = .PlotPhyloEnv)

# the original labels, and new (indexed) order of pedons in dendrogram
d.labels <- attr(d, 'Labels')

new_order <- sapply(1:lastPP$Ntip,
function(i) which(as.integer(lastPP$xx[1:lastPP$Ntip]) == i))

# plot the profiles, in the ordering defined by the dendrogram
# with a couple fudge factors to make them fit
plot(sp3.grouped, color="soil_color", plot.order=new_order,
scaling.factor=0.01, width=0.1, cex.names=0.5,
y.offset=max(lastPP$yy)+0.1, add=TRUE)

}

```

Description

Soil Chemical Data from Serpentinic Soils of California

Usage

```
data(sp4)
```

Format

A data frame with 30 observations on the following 13 variables.

```
id site name
name horizon designation
top horizon top boundary in cm
bottom horizon bottom boundary in cm
K exchangeable K in c mol/kg
Mg exchangeable Mg in cmol/kg
Ca exchangeable Ca in cmol/kg
CEC_7 cation exchange capacity (NH4OAc at pH 7)
ex_Ca_to_Mg extractable Ca:Mg ratio
sand sand content by weight percentage
silt silt content by weight percentage
clay clay content by weight percentage
CF >2mm fraction by volume percentage
```

Details

Selected soil physical and chemical data from (McGahan et al., 2009).

Source

<https://www.soils.org/publications/sssaj/articles/73/6/2087>

References

McGahan, D.G., Southard, R.J, Claassen, V.P. 2009. Plant-Available Calcium Varies Widely in Soils on Serpentine Landscapes. Soil Sci. Soc. Am. J. 73: 2087-2095.

Examples

```
# setup environment
library(aqp)

# load sample data set, a simple data.frame object with horizon-level data from 10 profiles
data(sp4)
str(sp4)

# optionally read about it...
# ?sp4

# upgrade to SoilProfileCollection
```

```

# 'id' is the name of the column containing the profile ID
# 'top' is the name of the column containing horizon upper boundaries
# 'bottom' is the name of the column containing horizon lower boundaries
depths(sp4) <- id ~ top + bottom

# check it out
class(sp4) # class name
str(sp4) # internal structure

# inspect object properties
idname(sp4) # self-explanatory
horizonDepths(sp4) # self-explanatory

# you can change these:
depth_units(sp4) # defaults to 'cm'
metadata(sp4) # not much to start with

# alter the depth unit metadata
depth_units(sp4) <- 'inches' # units are really 'cm'

# more generic interface for adjusting metadata
md <- metadata(sp4) # save original metadata

# add columns
md$describer <- 'DGM'
md$date <- as.Date('2009-01-01')
md$citation <- 'McGahan, D.G., Southard, R.J, Claassen, V.P.
2009. Plant-Available Calcium Varies Widely in Soils
on Serpentine Landscapes. Soil Sci. Soc. Am. J. 73: 2087-2095.'

# re-assign
metadata(sp4) <- md
depth_units(sp4) <- 'cm' # fix depth units, back to 'cm'

# further inspection with common function overloads
length(sp4) # number of profiles in the collection
nrow(sp4) # number of horizons in the collection
names(sp4) # column names
min(sp4) # shallowest profile depth in collection
max(sp4) # deepest profile depth in collection

# extraction of soil profile components
profile_id(sp4) # vector of profile IDs
horizons(sp4) # horizon data

# extraction of specific horizon attributes
sp4$clay # vector of clay content

# subsetting SoilProfileCollection objects
sp4[1, ] # first profile in the collection
sp4[, 1] # first horizon from each profile

# basic plot method, highly customizable: see manual page ?plotSPC

```

```

plot(sp4)
# inspect plotting area, very simple to overlay graphical elements
abline(v=1:length(sp4), lty=3, col='blue')
# profiles are centered at integers, from 1 to length(obj)
axis(1, line=-1.5, at=1:10, cex.axis=0.75, font=4, col='blue', lwd=2)
# y-axis is based on profile depths
axis(2, line=-1, at=pretty(1:max(sp4)), cex.axis=0.75, font=4, las=1, col='blue', lwd=2)

# symbolize soil properties via color
par(mar=c(0,0,4,0))
plot(sp4, color='clay')
plot(sp4, color='CF')

# apply a function to each profile, returning a single value per profile,
# in the same order as profile_id(sp4)
soil.depths <- profileApply(sp4, max) # recall that max() gives the depth of a soil profile

# check that the order is correct
all.equal(names(soil.depths), profile_id(sp4))

# a vector of values that is the same length as the number of profiles
# can be stored into site-level data
sp4$depth <- soil.depths
# check: looks good
max(sp4[,1]) == sp4$depth[1]

# extract site-level data
site(sp4) # as a data.frame
sp4$depth # specific columns as a vector

# use site-level data to alter plotting order
new.order <- order(sp4$depth) # the result is an index of rank
par(mar=c(0,0,0,0))
plot(sp4, plot.order=new.order)

# deconstruct SoilProfileCollection into a data.frame, with horizon+site data
as(sp4, 'data.frame')

```

sp5

Sample Soil Database #5

Description

296 Soil Profiles from the La Rochelle region of France (F. Carre and Girard, 2002)

Usage

```
data(sp5)
```

Format

```

Formal class 'SoilProfileCollection' [package "aqp"] with 6 slots
..@ idcol      : chr "soil"
..@ depthcols: chr [1:2] "top" "bottom"
..@ metadata  :'data.frame': 1 obs. of  1 variable:
.. ..$ depth_units: chr "cm"
..@ horizons  :'data.frame': 1539 obs. of  17 variables:
.. ..$ soil      : soil ID
.. ..$ sand      : sand
.. ..$ silt      : silt
.. ..$ clay      : clay
.. ..$ R25       : RGB r-coordinate
.. ..$ G25       : RGB g-coordinate
.. ..$ B25       : RGB b-coordinate
.. ..$ pH        : pH
.. ..$ EC        : EC
.. ..$ CaC03     : CaC03 content
.. ..$ C         : C content
.. ..$ Ca        : Ca
.. ..$ Mg        : Mg
.. ..$ Na        : Na
.. ..$ top       : horizon top boundary (cm)
.. ..$ bottom    : horizon bottom boundary (cm)
.. ..$ soil_color: soil color in r-friendly format
..@ site       :'data.frame': 296 obs. of  1 variable:
.. ..$ soil: chr [1:296] "soil1" "soil10" "soil100" "soil101" ...
..@ sp         :Formal class 'SpatialPoints' [package "sp"] with 3 slots
.. .. ..@ coords      : num [1, 1] 0
.. .. ..@ bbox        : logi [1, 1] NA
.. .. ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
.. .. .. ..@ projargs: chr NA

```

Details

These data are c/o F. Carre (Florence.CARRE@ineris.fr).

Source

296 Soil Profiles from the La Rochelle region of France (F. Carre and Girard, 2002). These data can be found on the OSACA project page (<http://eusoils.jrc.ec.europa.eu/projects/OSACA/>).

References

F. Carre, M.C. Girard. 2002. Quantitative mapping of soil types based on regression kriging of taxonomic distances with landform and land cover attributes. *Geoderma*. 110: 241–263.

Examples

```
library(scales)
```

```

data(sp5)
par(mar=c(1,1,1,1))
# plot a random sampling of profiles
s <- sample(1:length(sp5), size=25)
plot(sp5[s, ], divide.hz=FALSE)

# plot the first 100 profiles, as 4 rows of 25, hard-coding the max depth
layout(matrix(c(1,2,3,4), ncol=1), height=c(0.25,0.25,0.25,0.25))
plot(sp5[1:25, ], max.depth=300)
plot(sp5[26:50, ], max.depth=300)
plot(sp5[51:75, ], max.depth=300)
plot(sp5[76:100, ], max.depth=300)

# 4x1 matrix of plotting areas
layout(matrix(c(1,2,3,4), ncol=1), height=c(0.25,0.25,0.25,0.25))

# plot profiles, with points added to the mid-points of randomly selected horizons
sub <- sp5[1:25, ]
plot(sub, max.depth=300) ; mtext('Set 1', 2, line=-0.5, font=2)
y.p <- profileApply(sub, function(x) {
  s <- sample(1:nrow(x), 1)
  h <- horizons(x); with(h[s,], (top+bottom)/2)
})
points(1:25, y.p, bg='white', pch=21)

# plot profiles, with arrows pointing to profile bottoms
sub <- sp5[26:50, ]
plot(sub, max.depth=300); mtext('Set 2', 2, line=-0.5, font=2)
y.a <- profileApply(sub, function(x) max(x))
arrows(1:25, y.a-50, 1:25, y.a, len=0.1, col='white')

# plot profiles, with points connected by lines: ideally reflecting some kind of measured data
sub <- sp5[51:75, ]
plot(sub, max.depth=300); mtext('Set 3', 2, line=-0.5, font=2)
y.p <- 20*(sin(1:25) + 2*cos(1:25) + 5)
points(1:25, y.p, bg='white', pch=21)
lines(1:25, y.p, lty=2)

# plot profiles, with polygons connecting horizons with max clay content (+/-) 10 cm
sub <- sp5[76:100, ]
y.clay.max <- profileApply(sub, function(x) {
  i <- which.max(x$clay)
  h <- horizons(x)
  with(h[i, ], (top+bottom)/2)
})

plot(sub, max.depth=300); mtext('Set 4', 2, line=-0.5, font=2)
polygon(c(1:25, 25:1), c(y.clay.max-10, rev(y.clay.max+10)),
border='black', col=rgb(0,0,0.8, alpha=0.25))
points(1:25, y.clay.max, pch=21, bg='white')

# close plot

```

```

dev.off()

# plotting parameters
yo <- 100 # y-offset
sf <- 0.65 # scaling factor
# plot profile sketches
par(mar=c(0,0,0,0))
plot(sp5[1:25, ], max.depth=300, y.offset=yo, scaling.factor=sf)
# optionally add describe plotting area above profiles with lines
# abline(h=c(0,90,100, (300*sf)+yo), lty=2)
# simulate an environmental variable associated with profiles (elevation, etc.)
r <- vector(mode='numeric', length=25)
r[1] <- -50 ; for(i in 2:25) {r[i] <- r[i-1] + rnorm(mean=-1, sd=25, n=1)}
# rescale
r <- rescale(r, to=c(80, 0))
# illustrate gradient with points/lines/arrows
lines(1:25, r)
points(1:25, r, pch=16)
arrows(1:25, r, 1:25, 95, len=0.1)
# add scale for simulated gradient
axis(2, at=pretty(0:80), labels=rev(pretty(0:80)), line=-1, cex.axis=0.75, las=2)
# depict a secondary environmental gradient with polygons (water table depth, etc.)
polygon(c(1:25, 25:1), c((100-r)+150, rep((300*sf)+yo, times=25)),
border='black', col=rgb(0,0,0.8, alpha=0.25))

##
# sample 25 profiles from the collection
s <- sp5[sample(1:length(sp5), size=25), ]
# compute pair-wise dissimilarity
d <- profile_compare(s, vars=c('R25', 'pH', 'clay', 'EC'), k=0,
replace_na=TRUE, add_soil_flag=TRUE, max_d=300)
# keep only the dissimilarity between profile 1 and all others
d.1 <- as.matrix(d)[1, ]
# rescale dissimilarities
d.1 <- rescale(d.1, to=c(80, 0))
# sort in ascending order
d.1.order <- rev(order(d.1))
# plotting parameters
yo <- 100 # y-offset
sf <- 0.65 # scaling factor
# plot sketches
par(mar=c(0,0,0,0))
plot(s, max.depth=300, y.offset=yo, scaling.factor=sf, plot.order=d.1.order)
# add dissimilarity values with lines/points
lines(1:25, d.1[d.1.order])
points(1:25, d.1[d.1.order], pch=16)
# link dissimilarity values with profile sketches via arrows
arrows(1:25, d.1[d.1.order], 1:25, 95, len=0.1)
# add an axis for the dissimilarity scale
axis(2, at=pretty(0:80), labels=rev(pretty(0:80)), line=-1, cex.axis=0.75, las=2)

```

sp6

Soil Physical and Chemical Data from Manganiferous Soils

Description

Soil Physical and Chemical Data from Manganiferous Soils (Bourgault and Rabenhorst, 2011)

Usage

data(sp6)

Format

A data frame with 30 observations on the following 13 variables.

id pedon name
name horizon designation
top horizon top boundary in cm
bottom horizon bottom boundary in cm
color moist soil color in Munsell notation
texture USDA soil texture class
sand sand content by weight percentage
silt silt content by weight percentage
clay clay content by weight percentage
Fe DCB-extracted Fe in g/kg (see citation)
Mn DCB-extracted Mn in g/kg (see citation)
C total organic carbon as g/kg
pH measured in 1:1 H₂O slurry
Db bulk density (g/cc), clod method

Details

Selected soil physical and chemical data from (Bourgault and Rabenhorst, 2011).

Source

<http://www.sciencedirect.com/science/article/pii/S0016706111001972>

References

Rebecca R. Bourgault, Martin C. Rabenhorst. 2011. Genesis and characterization of manganiferous soils in the Eastern Piedmont, USA. *Geoderma*. 165:84-94.

Examples

```

# setup environment
library(aqp)
data(sp6)

# init SPC
depths(sp6) <- id ~ top + bottom
# convert non-standard Munsell colors
sp6$soil_color <- getClosestMunsellChip(sp6$color)

# profile sketches
par(mar=c(0,0,3,0))
plot(sp6, color='soil_color')
plot(sp6, color='Mn')
plot(sp6, color='Fe')
plot(sp6, color='pH')
plot(sp6, color='texture')

```

SPC-utils

Getters, Setters, and Utility Methods for SoilProfileCollection Objects

Description

Getters, Setters, and Utility Methods for SoilProfileCollection Objects

Methods

```
signature(object = "SoilProfileCollection")
```

Author(s)

Dylan E. Beaudette

References

<http://casoilresource.lawr.ucdavis.edu/>

Examples

```

data(sp1)

## init SoilProfileCollection objects from data.frame
depths(sp1) <- id ~ top + bottom

## depth units
(du <- depth_units(sp1))
depth_units(sp1) <- 'in'
depth_units(sp1) <- du

```

```
## get/set metadata on SoilProfileCollection objects
# this is a 1-row data.frame
m <- metadata(sp1)
m$sampler <- 'Dylan'
metadata(sp1) <- m

## extract horizon data from SoilProfileCollection objects as data.frame
h <- horizons(sp1)

# also replace horizon data in SoilProfileCollection objects
# original order and length must be preserved!
horizons(sp1) <- h

# get number of horizons
nrow(sp1)

## getting site-level data
site(sp1)

## setting site-level data
# site-level data from horizon-level data (stored in @horizons)
site(sp1) <- ~ group

# make some fake site data, and append from data.frame
# a matching ID column must be present in both @site and new data
# note that IDs should all be character class
d <- data.frame(id=profile_id(sp1), p=runif(n=length(sp1)), stringsAsFactors=FALSE)
site(sp1) <- d

# edit horizon depths
horizonDepths(sp1) <- c('t', 'b')
horizonDepths(sp1)

# edit profile IDs
p <- sprintf("%s-new", profile_id(sp1))
profile_id(sp1) <- p
profile_id(sp1)
```

subsetProfiles-methods

Subset SoilProfileCollection Objects.

Description

This function is used to subset SoilProfileCollection objects using either site-level or horizon-level attributes, or both.

Details

The `s` argument supplies a fully-quoted search criteria for matching via site-level attributes. The `h` argument supplies a fully-quoted search criteria for matching via horizon-level attributes. All horizons associated with a single horizon-level match (i.e. out of several, only a single horizon matches the search criteria) are returned. See examples for usage.

Value

A `SoilProfileCollection` class object.

Methods

```
signature(object = "SoilProfileCollection", s = 'character', h = 'character', ...)
```

See Also

[profileApply](#), [site](#), [horizons](#)

Examples

```
# more interesting sample data
data(sp2)
depths(sp2) <- id ~ top + bottom
site(sp2) <- ~ surface

# subset by integer index, note that this does not re-order the profiles
plot(sp2[1:5, ])

# generate an integer index via pattern-matching
idx <- grep('modesto', sp2$surface, ignore.case=TRUE)
plot(sp2[idx, ])

# generate in index via profileApply:
# subset those profiles where: min(ph) < 5.6
idx <- which(profileApply(sp2, function(i) min(i$field_ph, na.rm=TRUE) < 5.6))
plot(sp2[idx, ])
```

tau

Compute weighted naïve and tau statistics for a cross-classification matrix

Description

`tauW`: Computes: (1) unweighted naïve, (2) weighted naïve, (3) unweighted *tau*, (4) weighted *tau* accuracy statistics

`summaryTauW`: prints a summary of the results from *tauW*

`xtableTauW`: formats a LaTeX table with results from *tauW* and saves it as a `.tex` file for import into a LaTeX document.

Usage

```
tauW(CM, W = diag(sqrt(length(as.matrix(CM)))),
P = rep(1/nrow(as.matrix(CM)), nrow(as.matrix(CM))))

summaryTauW(result.tau)

xtableTauW(result.tau, file.name="tau_results_table.tex")
```

Arguments

CM	a square confusion (cross-classification) matrix (rows: allocation, columns: reference)
W	weights: 1 on diagonals, [-1..1] off giving partial credit to this error
P	prior probability vector, length = number of rows/columns in CM and W
result.tau	results returned by tauW
file.name	output LaTeX file

Details

Input matrices CM and W may be in data.frame format and will be converted

Weights matrix W: 0 = no credit; 1 = full credit; -1 = maximum penalty/ If absent, default is no partial credit, i.e., unweighted.

Prior probabilities vector P: If absent, P are equal priors for each class. Special value $P = 0$ is interpreted as $P =$ column marginals.

Error checks: CM must be square; P must have correct number of classes and sum to 1 +/- 0.0001; W & CM must be conformable

Value

Results are returned in a list with obvious R names

Author(s)

D G Rossiter

References

- Rossiter, D. G., Zeng, R., & Zhang, G.-L. (2017). *Accounting for taxonomic distance in accuracy assessment of soil class predictions*. *Geoderma*, 292, 118–127. <https://doi.org/10.1016/j.geoderma.2017.01.012>
- Ma, Z. K., & Redmond, R. L. (1995). *Tau-coefficients for accuracy assessment of classification of remote-sensing data*. *Photogrammetric Engineering and Remote Sensing*, 61(4), 435–439.
- Naesset, E. (1996). *Conditional tau coefficient for assessment of producer's accuracy of classified remotely sensed data*. *ISPRS Journal of Photogrammetry and Remote Sensing*, 51(2), 91–98. [http://dx.doi.org/10.1016/0924-2716\(69\)00007-4](http://dx.doi.org/10.1016/0924-2716(69)00007-4)

Examples

```

# example confusion matrix
# rows: allocation (user's counts)
# columns: reference (producer's counts)
crossclass <- matrix(data=c(2,1,0,5,0,0,
                            1,74,2,1,3,6,
                            0,5,8,6,1,3,
                            6,1,3,91,0,0,
                            0,4,0,0,0,4,
                            0,6,2,2,4,38),
                    nrow=6, byrow=TRUE)
row.names(crossclass) <- c("OP", "SA", "UA", "UC", "AV", "AC")
colnames(crossclass) <- row.names(crossclass)

# build the weights matrix
# how much credit for a mis-allocation
weights <- matrix(data=c(1.00,0.05,0.05,0.15,0.05,0.15,
                        0.05,1.00,0.05,0.05,0.05,0.35,
                        0.05,0.05,1.00,0.20,0.15,0.15,
                        0.15,0.05,0.25,1.00,0.10,0.25,
                        0.05,0.10,0.15,0.10,1.00,0.15,
                        0.20,0.30,0.10,0.25,0.20,1.00),
                  nrow=6, byrow=TRUE)

# unweighted accuracy
summaryTauW(nnaive <- tauW(crossclass))

# unweighted tau with equal priors, equivalent to Foody (1992) modified Kappa
tauW(crossclass)$tau

# unweighted tau with user's = producer's marginals, equivalent to original kappa
(priors <- apply(crossclass, 2, sum)/sum(crossclass))
tauW(crossclass, P=priors)$tau

# weighted accuracy; tau with equal priors
summaryTauW(weighted <- tauW(crossclass, W=weights))

# weighted accuracy; tau with user's = producer's marginals
summaryTauW(tauW(crossclass, W=weights, P=priors))

# change in accuracy statistics weighted vs. non-weighted
(weighted$overall.weighted - weighted$overall.naive)
(weighted$user.weighted - weighted$user.naive)
(weighted$prod.weighted - weighted$prod.naive)

```

test_hz_logic

Test Horizon Logic

Description

Simple tests for horizon logic, based on a simple data.frame of ordered horizons.

Usage

```
test_hz_logic(i, topcol, bottomcol, strict = FALSE)
```

Arguments

<code>i</code>	a data.frame associated with a single soil profile, ordered by depth
<code>topcol</code>	character, giving the name of the column in <code>i</code> that describes horizon top depth
<code>bottomcol</code>	character, giving the name of the column in <code>i</code> that describes horizon bottom depth
<code>strict</code>	logical, should continuity tests be performed– i.e. for non-contiguous horizon boundaries

Details

By default, this function tests for NA and overlapping horizons. If any either are encountered, FALSE is returned.

Value

logical: TRUE -> pass, FALSE -> fail

Author(s)

D.E. Beaudette

References

<http://casoilresource.lawr.ucdavis.edu/>

See Also

[depths<-](#)

Examples

```
## simple example: just one profile
data(sp1)
depths(sp1) <- id ~ top + bottom
s <- horizons(sp1[[1, ]])

## check
# fails due to missing hz boundary
s$bottom[6] <- NA # missing horizon boundary, common on bottom-most hz
test_hz_logic(s, 'top', 'bottom', strict=FALSE)

# fails due to inconsistent hz boundary
s$bottom[3] <- 30 # inconsistent hz boundary
test_hz_logic(s, 'top', 'bottom', strict=TRUE)
```

```

## filtering bad data
## Not run:
# missing bottom horizons
x$hzn_bot[!is.na(x$hzn_top) & is.na(x$hzn_bot)] <- x$hzn_top[!is.na(x$hzn_top) & is.na(x$hzn_bot)]

# remove 0 horizons where top > bottom
bad.0.hz.idx <- which(x$hzn_top > x$hzn_bot)
if(length(bad.0.hz.idx) > 0)
x <- x[-bad.0.hz.idx, ]

## End(Not run)

## checking for bad data: do this before promoting to SoilProfileCollection object
library(plyr)
data(sp1)

# horizon logic can be tested via data.frame, at 2 levels of scrutiny:
ddply(sp1, 'id', test_hz_logic, topcol='top', bottomcol='bottom', strict=FALSE)
ddply(sp1, 'id', test_hz_logic, topcol='top', bottomcol='bottom', strict=TRUE)

```

textureTriangleSummary

Soil Texture Low-RV-High as Defined by Quantiles

Description

This function accepts soil texture components (sand, silt, and clay percentages) and plots a soil texture triangle with a "representative value" (point) and low-high region (polygon) defined by quantiles (estimated with `Hmisc::hdquantile`). Marginal quantiles of sand, silt, and clay are used to define the boundary of a low-high region that encloses a several likely soil texture classes based on the values in `ssc`. The default settings place the RV symbol at the texture defined by marginal medians of sand, silt, and clay. The default low-high region is defined by the 5th and 95th marginal percentiles of sand, silt, and clay.

Usage

```

textureTriangleSummary(ssc, p=c(0.05, 0.5, 0.95), delta=1,
  texture.names=FALSE, pop.rv.col='red', range.col='RoyalBlue',
  range.alpha=75, sim=FALSE, sim.n=1000, sim.rv.col='yellow',
  sim.col=grey(0.95), sim.alpha=150, legend.cex=0.75, ...)

```

Arguments

<code>ssc</code>	a matrix-like object with columns: 'sand', 'silt', 'clay', values are percentages that should add to 100.
<code>p</code>	percentiles defining 'low', 'representative value', and 'high'
<code>delta</code>	step-size used to form low-high region
<code>texture.names</code>	logical, should soil texture names be added to the figure?

pop.rv.col	the symbol color used to denote the population representative value on the texture triangle
range.col	color of the polygon enclosing the low-high region
range.alpha	transparency of the low-high range polygon (0-255)
sim	optional simulation of low-rv-high values based on a composition drawn from normal distributions, this requires the 'compositions' package
sim.n	number of simulated sand, silt, and clay values
sim.rv.col	the symbol color used to denote the simulated representative value on the texture triangle
sim.col	color of the simulated low-high range polygon
sim.alpha	transparency of the simulated low-high range polygon (0-255)
legend.cex	scaling factor for legend
...	further arguments passed to triax.points

Details

Simulated sand, silt, and clay values are based on sampling from a normal distribution as performed by `rnorm.acomp` in the 'compositions' package. The mean vector of the sand, silt, and clay values, along with covariance matrix derived from `ssc` are used to parametrize sampling.

Value

A high-level plot as generated by `soil.texture`.

Note

Simulation of sand, silt, and clay values requires the 'compositions' package. When using this function within `.Rmd`, be sure to explicitly load the 'compositions' package via `library` and use the chunk option `fig.keep='last'`. For some reason, the `soil.texture` function generates two figures when used within a `.Rmd`, hence the strange hack.

Author(s)

D.E. Beaudette

See Also

[triax.points](#), [soil.texture](#)

Examples

```
# sample data
data(loafercreek, package='soilDB')

# extract sand, silt, clay proportions
x <- na.omit(data.frame(sand=loafercreek$sand, silt=loafercreek$silt, clay=loafercreek$clay))

# test out the function
```

```

textureTriangleSummary(x, p=c(0.05, 0.5, 0.95))
textureTriangleSummary(x, p=c(0.25, 0.5, 0.75), range.col='darkgreen')

# simulate compositional data from source mean / var-covar matrix
if(require(compositions)) {
  # add simulated low-rv-high
  textureTriangleSummary(x, p=c(0.05, 0.5, 0.95), sim=TRUE)
}

```

union

Combine Multiple SoilProfileCollection Objects

Description

Safely combine multiple SoilProfileCollection objects that may not share the same internal structure.

Usage

```
union(spc = list(), method = "all", drop.spatial=FALSE)
```

Arguments

spc	a list of SoilProfileCollection objects
method	method ("all" or "intersection" [not yet implemented]) used to collect site and horizon level attributes, see details
drop.spatial	union operation ignores spatial data, useful when only some data have coordinates or incompatible CRS

Details

Method "all" returns all site and horizon level attributes, padded as needed with NA. Method "intersection" (not yet implemented) returns only those site and horizon level attributes that exist in all objects. Input data must share a common depth unit, and if spatial data are present, a common CRS and coordinate names. In the case of non-conformal @idname and/or @depthcols, the first SoilProfileCollection is used as a template.

Value

a new SoilProfileCollection object

Note

Previously, rbind.SoilProfileCollection would remove duplicates. This is no longer performed and an error will be generated due to non-unique profile IDs.

Author(s)

D.E. Beaudette and A.G. Brown

Examples

```

# example data
data(sp2, package = 'aqp')
depths(sp2) <- id ~ top + bottom
site(sp2) <- ~ surface

# copy pieces
x <- sp2[1:5, ]
y <- sp2[6:10, ]

# reset IDs and combine
profile_id(y) <- sprintf("%s-copy", profile_id(y))

# this should work
z <- union(list(x,y))

# check
plot(z)

## Not run:
library(plyr)

ids <- sprintf("%02d", 1:5)
x <- ldply(ids, random_profile, n=c(6, 7, 8), n_prop=1, method='LPP',
          lpp.a=5, lpp.b=15, lpp.d=5, lpp.e=5, lpp.u=25)

# promote to SPC and plot
depths(x ) <- id ~ top + bottom
plot(x, color='p1')

# slice and update IDs
y <- slice(x, 0:150 ~ .)
profile_id(y) <- sprintf("%s-sliced", profile_id(x))

# stack, note that @horizons is not the same in x and y
z <- union(list(x, y))

# label groups
z$g <- substr(profile_id(z), 1, 2)

par(mar=c(0,0,3,0))
groupedProfilePlot(z, groups = 'g', color='p1', group.name.offset = -10, divide.hz=FALSE, name='')

## End(Not run)

```

`unique-methods`*Get Indices to Unique Soil Profiles Within a Collection*

Description

This function returns a set of indices to a subset of profiles within a `SoilProfileCollection` object that are uniquely defined by a named set of horizon and site level attributes.

Usage

```
uniqueSPC(x, vars)
```

Arguments

<code>x</code>	a <code>SoilProfileCollection</code>
<code>vars</code>	a character vector naming those horizon and site level attributes that will be used to test for duplication

Details

Duplicates are identified via MD5 hash of select horizon and site level attributes.

Value

A vector of integer indices that can be used to subset unique profiles from the original `SoilProfileCollection` object.

Methods

```
signature(x = "SoilProfileCollection")
```

Author(s)

D.E. Beaudette

Examples

```
# use the digest library to detect duplicate data
data(sp1)
sp1$soil_color <- with(sp1, munsell2rgb(hue, value, chroma))

# upgrade to SoilProfileCollection
depths(sp1) <- id ~ top + bottom
site(sp1) <- ~ group

# make a copies
s.1 <- sp1
s.2 <- sp1
```

```

# update IDs in second copy
profile_id(s.2) <- sprintf('%s-copy', profile_id(s.2))

# union SPCs
s <- union(list(s.1, s.2))

# check
plot(s)

# digests are computed from horizon-level data only
# horizon boundaries and 'prop'
# result is an index of unique profiles

u <- unique(s, vars=c('top', 'bottom', 'prop'))

# compare with and without dupes:
# note subsetting of SoilProfileCollection
cbind(dupes=length(s), no.dupes=length(s[u, ]))

# get unique profile by index
s.unique <- s[u, ]

# unique data
plot(s.unique)

```

unroll

Unroll Genetic Horizons

Description

Generate a discretized vector of genetic horizons along a user-defined pattern.

Usage

```
unroll(top, bottom, prop, max_depth, bottom_padding_value = NA, strict=FALSE)
```

Arguments

top	vector of upper horizon boundaries, must be an integer
bottom	vector of lower horizon boundaries, must be an integer
prop	vector of some property to be "unrolled" over a regular sequence
max_depth	maximum depth to which missing data is padded with NA
bottom_padding_value	value to use when padding missing data
strict	should horizons be strictly checked for self-consistency? defaults to FALSE

Details

This function is used internally by several higher-level components of the *aqp* package. Basic error checking is performed to make sure that bottom and top horizon boundaries make sense. Note that the horizons should be sorted according to depth before using this function. The `max_depth` argument is used to specify the maximum depth of profiles within a collection, so that data from any profile shallower than this depth is padded with NA.

Value

a vector of "unrolled" property values

Author(s)

Dylan E. Beaudette

References

<http://casoilresource.lawr.ucdavis.edu/>

Examples

```
data(sp1)

# subset a single soil profile:
sp1.1 <- subset(sp1, subset=id == 'P001')

# demonstrate how this function works
x <- with(sp1.1, unroll(top, bottom, prop, max_depth=50))
plot(x, 1:length(x), ylim=c(90,0), type='b', cex=0.5)
```

Index

- *Topic **\textasciitildekw1**
 - addVolumeFraction, 6
- *Topic **array**
 - tau, 108
- *Topic **classes**
 - SoilProfileCollection-class, 85
- *Topic **datasets**
 - ca630, 12
 - munsell, 47
 - rruff.sample, 71
 - soil_minerals, 90
 - sp1, 92
 - sp2, 94
 - sp3, 96
 - sp4, 98
 - sp5, 101
 - sp6, 105
- *Topic **hplots**
 - groupedProfilePlot, 39
 - missingDataGrid, 46
 - plotMultipleSPC, 53
- *Topic **hplot**
 - panel.depth_function, 51
 - plot_distance_graph, 54
 - textureTriangleSummary, 112
- *Topic **manip**
 - aggregateColor, 7
 - aggregateSoilDepth, 9
 - argillic.clay.increase.depth, 10
 - brierScore, 11
 - clod.hz.ids, 16
 - colorQuantiles, 17
 - confusionIndex, 18
 - crit.clay.argillic, 19
 - estimatePSCS, 20
 - estimateSoilDepth, 22
 - evalGenHZ, 23
 - evalMissingData, 25
 - f.noise, 26
 - generalize.hz, 29
 - get.increase.depths, 30
 - get.increase.matrix, 31
 - get.ml.hz, 32
 - getArgillicBounds, 34
 - getSoilDepthClass, 35
 - getSurfaceHorizonDepth, 36
 - glom, 38
 - guessGenHzLevels, 41
 - hzDistinctnessCodeToOffset, 43
 - hzTransitionProbabilities, 44
 - munsell2rgb, 48
 - profile_compare-methods, 62
 - profileApply-methods, 57
 - random_profile, 66
 - resample.twotheta, 69
 - shannonEntropy, 72
 - sim, 73
 - slab-methods, 74
 - slice-methods, 81
 - soilColorSignature, 83
 - subsetProfiles-methods, 107
 - test_hz_logic, 110
 - union, 114
 - unique-methods, 116
 - unroll, 117
- *Topic **methods**
 - profile_compare-methods, 62
 - profileApply-methods, 57
 - slab-methods, 74
 - slice-methods, 81
 - SPC-utils, 106
 - subsetProfiles-methods, 107
 - unique-methods, 116
- *Topic **package**
 - aqp-package, 3
 - .lpp (random_profile), 66
 - [, SoilProfileCollection, ANY, ANY, ANY-method (SoilProfileCollection-class),

- 85
- [,SoilProfileCollection-method
(SoilProfileCollection-class),
85
- \$,SoilProfileCollection-method
(SoilProfileCollection-class),
85
- \$<-,SoilProfileCollection-method
(SoilProfileCollection-class),
85
- addBracket, 4, 89
- addDiagnosticBracket (addBracket), 4
- addVolumeFraction, 6
- aggregateColor, 7
- aggregateSoilDepth, 9
- aqp (aqp-package), 3
- aqp-package, 3
- aqp.env (aqp-package), 3
- argillic.clay.increase.depth, 10
- brierScore, 11
- ca630, 4, 12
- checkSPC, 15, 69
- clod.hz.ids, 16
- colorQuantiles, 17
- confusionIndex, 18
- coordinates,SoilProfileCollection-method
(SoilProfileCollection-class),
85
- coordinates<-,SoilProfileCollection-method
(SoilProfileCollection-class),
85
- create_progress_bar, 63
- crit.clay.argillic, 19
- daisy, 24, 64
- depth_units (SPC-utils), 106
- depth_units,SoilProfileCollection-method
(SPC-utils), 106
- depth_units<- (SPC-utils), 106
- depth_units<-,SoilProfileCollection-method
(SPC-utils), 106
- depths<-, 111
- depths<- (SPC-utils), 106
- depths<-,data.frame-method (SPC-utils),
106
- depths<-,SoilProfileCollection-method
(SPC-utils), 106
- diagnostic_hz (SPC-utils), 106
- diagnostic_hz,SoilProfileCollection-method
(SPC-utils), 106
- diagnostic_hz<- (SPC-utils), 106
- diagnostic_hz<-,SoilProfileCollection-method
(SPC-utils), 106
- estimatePSCS, 20
- estimateSoilDepth, 10, 22, 35, 36, 57
- evalGenHZ, 23
- evalMissingData, 25
- explainPlotSPC
(SoilProfileCollection-plotting-methods),
87
- f.noise, 26
- generalize.hz, 8, 29, 42, 45
- genhzTableToAdjMat
(hzTransitionProbabilities), 44
- genSlabLabels (slab-methods), 74
- get.increase.depths, 30
- get.increase.matrix, 20, 31
- get.ml.hz, 24, 32
- get.slice (slice-methods), 81
- getArgillicBounds, 20, 34
- getClosestMunsellChip (munsell2rgb), 48
- getMineralSoilSurfaceDepth
(getSurfaceHorizonDepth), 36
- getPlowLayerDepth
(getSurfaceHorizonDepth), 36
- getSoilDepthClass, 22, 35
- getSurfaceHorizonDepth, 36
- glom, 38
- groupedProfilePlot, 39
- guessGenHzLevels, 41
- hdquantile, 75, 77
- horizonDepths (SPC-utils), 106
- horizonDepths,SoilProfileCollection-method
(SPC-utils), 106
- horizonDepths<- (SPC-utils), 106
- horizonDepths<-,SoilProfileCollection-method
(SPC-utils), 106
- horizonNames
(SoilProfileCollection-class),
85
- horizonNames,SoilProfileCollection-method
(SoilProfileCollection-class),
85

- horizonNames<-
 (SoilProfileCollection-class),
 85
- horizonNames<-, SoilProfileCollection-method
 (SoilProfileCollection-class),
 85
- horizons, 108
- horizons (SPC-utils), 106
- horizons, SoilProfileCollection-method
 (SPC-utils), 106
- horizons<- (SPC-utils), 106
- horizons<-, SoilProfileCollection-method
 (SPC-utils), 106
- hzDistinctnessCodeToOffset, 43, 67, 89
- hzID (SPC-utils), 106
- hzID, SoilProfileCollection-method
 (SPC-utils), 106
- hzID<- (SPC-utils), 106
- hzID<-, SoilProfileCollection-method
 (SPC-utils), 106
- hzidname (SPC-utils), 106
- hzidname, SoilProfileCollection-method
 (SPC-utils), 106
- hzidname<- (SPC-utils), 106
- hzidname<-, SoilProfileCollection-method
 (SPC-utils), 106
- hzTransitionProbabilities, 44
- idname (SPC-utils), 106
- idname, SoilProfileCollection-method
 (SPC-utils), 106
- isoMDS, 24
- length, SoilProfileCollection-method
 (SoilProfileCollection-class),
 85
- make.segments (panel.depth_function), 51
- max, SoilProfileCollection-method
 (SoilProfileCollection-class),
 85
- metadata (SPC-utils), 106
- metadata, SoilProfileCollection-method
 (SPC-utils), 106
- metadata<- (SPC-utils), 106
- metadata<-, SoilProfileCollection-method
 (SPC-utils), 106
- min, SoilProfileCollection-method
 (SoilProfileCollection-class),
 85
- missingDataGrid, 46
- mostLikelyHzSequence
 (hzTransitionProbabilities), 44
- munsell, 47
- munsell2rgb, 48, 84, 89
- names (SoilProfileCollection-class), 85
- names, SoilProfileCollection-method
 (SoilProfileCollection-class),
 85
- nrow, SoilProfileCollection-method
 (SoilProfileCollection-class),
 85
- panel.depth_function, 51
- parseMunsell (munsell2rgb), 48
- pc (profile_compare-methods), 62
- plot
 (SoilProfileCollection-plotting-methods),
 87
- plot, SoilProfileCollection, ANY-method
 (SoilProfileCollection-plotting-methods),
 87
- plot, SoilProfileCollection-method
 (SoilProfileCollection-plotting-methods),
 87
- plot.SoilProfileCollection
 (SoilProfileCollection-plotting-methods),
 87
- plot_distance_graph, 54
- plotColorQuantiles (colorQuantiles), 17
- plotMultipleSPC, 53, 61
- plotSPC, 5, 6, 40, 43
- plotSPC
 (SoilProfileCollection-plotting-methods),
 87
- prepanel.depth_function
 (panel.depth_function), 51
- pretty, 88, 89
- previewColors, 55
- profile_compare, 55, 63, 67
- profile_compare
 (profile_compare-methods), 62
- profile_compare, data.frame-method
 (profile_compare-methods), 62
- profile_compare, SoilProfileCollection-method
 (profile_compare-methods), 62
- profile_compare-methods, 62
- profile_id (SPC-utils), 106

- profile_id, SoilProfileCollection-method (SPC-utils), 106
- profile_id<- (SPC-utils), 106
- profile_id<-, SoilProfileCollection-method (SPC-utils), 106
- profileApply, 22, 108
- profileApply (profileApply-methods), 57
- profileApply, SoilProfileCollection-method (profileApply-methods), 57
- profileApply-methods, 57
- profileGroupLabels, 54, 60, 89
- proj4string, SoilProfileCollection-method (SoilProfileCollection-class), 85
- proj4string<-, SoilProfileCollection, ANY-method (SoilProfileCollection-class), 85
- random_profile, 66, 73
- rbind. SoilProfileCollection (SoilProfileCollection-class), 85
- rebuildSPC, 15, 69
- resample.twotheta, 26, 69
- rgb, 48
- rgb2munsell (munsell2rgb), 48
- rruff.sample, 70, 71
- shannonEntropy, 72
- show, SoilProfileCollection-method (SoilProfileCollection-class), 85
- silhouette, 24
- sim, 73
- site, 108
- site (SPC-utils), 106
- site, SoilProfileCollection-method (SPC-utils), 106
- site<- (SPC-utils), 106
- site<-, SoilProfileCollection-method (SPC-utils), 106
- siteNames (SoilProfileCollection-class), 85
- siteNames, SoilProfileCollection-method (SoilProfileCollection-class), 85
- siteNames<- (SoilProfileCollection-class), 85
- siteNames<-, SoilProfileCollection-method (SoilProfileCollection-class), 85
- slab, 10, 32, 33, 39, 52, 57, 82
- slab (slab-methods), 74
- slab, SoilProfileCollection-method (slab-methods), 74
- slab-methods, 74
- slab2 (slab-methods), 74
- slice, 39, 47, 52, 64, 77
- slice (slice-methods), 81
- slice, SoilProfileCollection-method (slice-methods), 81
- slice-methods, 81
- slice.fast (slice-methods), 81
- soil.texture, 113
- soil_minerals, 90
- soilColorSignature, 83
- SoilProfileCollection, 62, 63, 76, 81
- SoilProfileCollection (SoilProfileCollection-class), 85
- SoilProfileCollection-class, 85
- SoilProfileCollection-plotting-methods, 87
- sp1, 4, 52, 92
- sp2, 4, 55, 94
- sp3, 4, 96
- sp4, 4, 98
- sp5, 4, 101
- sp6, 105
- SPC-utils, 106
- subsetProfiles (subsetProfiles-methods), 107
- subsetProfiles, SoilProfileCollection-method (subsetProfiles-methods), 107
- subsetProfiles-methods, 107
- summaryTauW (tau), 108
- tau, 108
- tauW (tau), 108
- test_hz_logic, 110
- texture.triangle.low.rv.high (textureTriangleSummary), 112
- textureTriangleSummary, 112
- triax.points, 113
- union, 114

unique (unique-methods), [116](#)
unique, SoilProfileCollection-method
 (unique-methods), [116](#)
unique-methods, [116](#)
uniqueSPC (unique-methods), [116](#)
unroll, [117](#)

xtableTauW (tau), [108](#)