

Package ‘aqp’

April 8, 2021

Version 1.29

Date 2021-04-05

Title Algorithms for Quantitative Pedology

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

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

Depends R (>= 3.5.0)

Imports grDevices, graphics, stats, utils, methods, plyr, grid,
lattice, cluster, sp, stringr, data.table

Suggests colorspace, ape, soilDB, latticeExtra, tactile, compositions,
sharpshootR, markovchain, xtable, testthat, Gmedian, farver,
Hmisc, tibble, RColorBrewer, scales, digest, MASS, mpspline2,
soiltexture, gower

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 7.1.1

NeedsCompilation no

Date/Publication 2021-04-07 22:50:02 UTC

R topics documented:

aqp-package	6
.as.data.frame.aqp	7
.data_dots	7
.HSD	8
.makeEquivalentMunsellLUT	8
.parseGrouped_formula	10
addBracket	10
addDiagnosticBracket	12
addVolumeFraction	13
aggregateColor	14
aggregateSoilDepth	17
alignTransect	18
allocate	19
aqp_df_class	23
argillic.clay.increase.depth	24
as	25
barron.torrent.redness.LAB	26
bootstrapSoilTexture	26
brierScore	29
buntley.westin.index	30
c,SoilProfileCollection-method	31
ca630	32
checkHzDepthLogic	35
checkSPC	37
colorContrast	38
colorContrastPlot	39
colorQuantiles	41
compositeSPC	42
confusionIndex	43
contrastChart	44
contrastClass	45
coordinates	46
correctAWC	47
crit.clay.argillic	48
denormalize	49
depthOf	50
depths<- ,SoilProfileCollection-method	52

depthWeights	53
depth_units	53
diagnostic_hz	54
diagnostic_hz<-	54
dice	55
duplicate	56
equivalentMunsellChips	57
equivalent_munsell	59
estimateAWC	60
estimatePSCS	61
estimateSoilDepth	63
evalGenHZ	65
evalMissingData	67
explainPlotSPC	68
f.noise	69
fillHzGaps	72
findOverlap	74
fixOverlap	75
generalize_hz	77
get.increase.depths	79
get.increase.matrix	80
get.ml_hz	81
getArgillicBounds	83
getCambicBounds	85
getClosestMunsellChip	86
getLastHorizonID	87
getSoilDepthClass	87
getSurfaceHorizonDepth	88
glom,SoilProfileCollection-method	90
glomApply	92
grepSPC	93
groupedProfilePlot	94
groupSPC	97
guessGenHzLevels	97
guessHzAttrName	99
guessHzDesgnName	100
guessHzTexCIName	101
harden.melanization	102
harden.rubification	104
harmonize,SoilProfileCollection-method	106
hasDarkColors	108
horizonColorIndices	110
horizonDepths<-	111
horizonNames<-	111
horizons	112
huePosition	112
hurst.redness	114
HzDepthLogicSubset	114

hzDepthTests	115
hzDesgn	116
hzdesgnname	116
hzdesgnname<-	117
hzDistinctnessCodeToOffset	117
hzID	119
hzID<-	119
hzidname<-	119
hztexclname	120
hztexclname<-	120
hzTopographyCodeToLineType	121
hzTopographyCodeToOffset	122
hzTransitionProbabilities	123
idname	125
invertLabelColor	125
jacobs2000	126
L1_profiles	127
length	128
lunique	129
max	130
metadata	130
min	131
missingDataGrid	132
mixMunsell	133
mollic.thickness.requirement	135
munsell	136
munsell.spectra	138
munsell2rgb	139
munsell2spc,SoilProfileCollection-method	141
mutate_profile	143
names	143
nrow	144
overlapMetrics	144
panel.depth_function	145
parseMunsell	149
pbindlist	150
pc	151
perturb	155
plotColorMixture	159
plotColorQuantiles	160
plotMultipleSPC	160
plotSPC	164
plot_distance_graph	171
pms.munsell.lut	172
previewColors	174
profileApply	176
profileGroupLabels	179
profile_id<-	181

proj4string,SoilProfileCollection-method	181
proj4string<-,SoilProfileCollection,ANY-method	181
random_profile	182
rebuildSPC	185
reorderHorizons	186
repairMissingHzDepths	186
replaceHorizons<-	187
resample.twotheta	187
restrictions	189
restrictions<-	189
rgb2munsell	190
ROSETTA.centroids	191
rowley2019	193
rruff.sample	196
segment	197
shannonEntropy	199
show	200
sierraTransect	201
sim	203
site	204
siteNames<-	205
slab-methods	205
slice-methods	214
slicedHSD	216
soilColorSignature	217
soilPalette	219
SoilProfileCollection	220
soiltexture	223
SoilTextureLevels	224
soil_minerals	225
sp1	227
sp2	228
sp3	230
sp4	233
sp5	236
sp6	239
spc2mpspline,SoilProfileCollection-method	240
spc_in_sync	241
split,SoilProfileCollection-method	242
splitLogicErrors	243
subApply	245
subset,SoilProfileCollection-method	245
subsetProfiles	246
summarizeSPC	247
tauW	248
texcl_to_ssc	250
textureTriangleSummary	253
thompson.bell.darkness	255

transform,SoilProfileCollection-method	256
trunc,SoilProfileCollection-method	257
unique	258
unroll	259
us.state.soils	260
validSpatialData	260
xtableTauW	261
[,SoilProfileCollection-method	261
[[.	262
[[<-	263
\$	264
\$<-	264

Index	266
--------------	------------

aqp-package

Algorithms for Quantitative Pedology

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.

Usage

aqp.env

Format

An object of class environment of length 0.

Details

Demos: `demo(aqp)`

[Project homepage](#)

Author(s)

Dylan E. Beaudette debeaudette@ucdavis.edu, Pierre Roudier, Andrew G. Brown

See Also

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

<code>.as.data.frame.aqp</code>	<i>Wrapper method for data.frame subclass conversion</i>
---------------------------------	--

Description

Wrapper method for data.frame subclass conversion

Usage

```
.as.data.frame.aqp(x, as.class = "data.frame", ...)
```

Arguments

<code>x</code>	ANY.
<code>as.class</code>	"data.frame", "tibble", or "data.table" default: "data.frame"
<code>...</code>	Additional arguments to coercion function <code>as.data.frame</code> , <code>as_tibble</code> or <code>as.data.table</code>

Value

a subclass of `data.frame` corresponding to `as.class`,

<code>.data_dots</code>	<i>Make a data.frame from non-standard expressions evaluated in a data environment</i>
-------------------------	--

Description

Make a data.frame from non-standard expressions evaluated in a data environment

Usage

```
.data_dots(.data, ...)
```

Arguments

<code>.data</code>	A list, or object coercible to one, describing the data
<code>...</code>	One or more expressions (preferably named e.g. <code>foo = "bar"</code>) to evaluate in <code>.data</code>

Value

A list where names are expression "names" from `...` and values are the result of evaluating expressions in context of `.data`

Examples

```
# .data_dots(data.frame(a = 1:10, b = 2:11), cc = a + b, d = cc * 2)
# data("jacobs2000", package="aqp")
# .data_dots(compositeSPC(jacobs2000), clayprop = clay / 100)
```

`.HSD`

.HSD

Description

Safely compute Tukey's HSD within a given slice

Usage

```
.HSD(z, aov.fm, conf = 0.95)
```

Arguments

<code>z</code>	data.frame containing basic metadata, horizon top/bottom, variable of interest, and grouping variable
<code>aov.fm</code>	formula suitable for <code>aov()</code> : <code>variable ~ group</code>
<code>conf</code>	confidence level for TukeyHSD

`.makeEquivalentMunsellLUT`

.makeEquivalentMunsellLUT

Description

Makes the look up table based on pair-wise CIE2000 color contrast (dE_{00}) of LAB colors with D65 illuminant of LAB colors for all whole value/chroma "chips" in the `aqp::munsell` data set via `farver::compare_colour`. Specify a threshold in terms of a probability level of CIE2000 distance (relative to whole dataset).

Usage

```
.makeEquivalentMunsellLUT(threshold = 0.001)
```

Arguments

<code>threshold</code>	Quantile cutoff (of dE_{00} color contrast) for "equivalent" colors, default <code>0.001</code> based on all whole value/chroma "chips" in the <code>munsell</code> data set.
------------------------	---

Value

A list with equal length to the number of rows (chips) in `munsell` data, each containing a numeric vector of row indices that are equivalent (CIE2000 distance less than threshold).

References

Gaurav Sharma, Wencheng Wu, Edul N. Dalal. (2005). The CIEDE2000 Color-Difference Formula: Implementation Notes, Supplementary Test Data, and Mathematical Observations. *COLOR research and application*. 30(1):21-30. <http://www2.ece.rochester.edu/~gsharma/ciede2000/ciede2000noteCRNA.pdf>

Thomas Lin Pedersen, Berendea Nicolae and Romain François (2020). `farver`: High Performance Colour Space Manipulation. R package version 2.0.3. <https://CRAN.R-project.org/package=farver>

Dong, C.E., Webb, J.B., Bottrell, M.C., Saginor, I., Lee, B.D. and Stern, L.A. (2020). Strengths, Limitations, and Recommendations for Instrumental Color Measurement in Forensic Soil Characterization. *J Forensic Sci*, 65: 438-449. <https://doi.org/10.1111/1556-4029.14193>

See Also

[equivalentMunsellChips](#)

Examples

```
## Not run:

# use default threshold
equivalent_chip_lut <- .makeEquivalentMunsellLUT()

# inspect the first 10 chips, it seems to work!
lapply(equivalent_chip_lut[1:10], function(i) munsell[i,])

# lets see some info on the number of chips per chip
nchipsper <- sapply(equivalent_chip_lut, length)

# top 10 are very high chroma chips with over 70 chips "identical"
nchipsper[order(nchipsper, decreasing = TRUE)[1:10]]

# look at distribution
plot(density(nchipsper))

# median is 5 -- Q: is this true of the range of Munsell colors typically used for soils?
quantile(nchipsper)

# double the default threshold
doubletest <- sapply(.makeEquivalentMunsellLUT(threshold = 0.002), length)
lines(density(doubletest), lty=2)

# apprx. doubles the number of chips per chip in IQR
quantile(doubletest)

## End(Not run)
```

```
.parseGrouped_formula .parseHSD_formula
```

Description

internally used function to parse slicedHSD formula notation

Usage

```
.parseGrouped_formula(fm)
```

Arguments

fm to parse, 0:100 ~ variable | group

Value

list with formula pieces

```
addBracket                    Add Depth Brackets
```

Description

Add depth brackets to soil profile sketches.

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 object containing idname(x), 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 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

x may contain multiple records per profile. 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

[addDiagnosticBracket](#), [plotSPC](#)

Examples

```

# 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
par(mar = c(0, 0, 0, 1))
plotSPC(sp1, width = 0.3)

# extract min--max depths associated with all A horizons
# result is a single-row data.frame / profile
combinedBracket <- function(i) {
  h <- horizons(i)
  idn <- idname(i)
  this.id <- h[[idn]][1]

  idx <- grep('^A', h$name)

  res <- data.frame(
    id = this.id,
    top = min(h$top[idx]),
  
```

```

    bottom = max(h$bottom[idx], na.rm=TRUE)
  )
  names(res)[1] <- idn

  return(res)
}

# return matching horizon top / bottom depths for A or C horizons
# result is a 0 or more row data.frame / profile
individualBrackets <- function(i) {
  h <- horizons(i)
  idn <- idname(i)
  this.id <- h[[idn]][1]

  idx <- grep('^A|^C', h$name)

  res <- data.frame(
    id = this.id,
    top = h$top[idx],
    bottom = h$bottom[idx]
  )
  names(res)[1] <- idn

  return(res)
}

# combined brackets
b1 <- profileApply(sp1, combinedBracket, frameify = TRUE)

# individual brackets
b2 <- profileApply(sp1, individualBrackets, frameify = TRUE)

# plot in reverse order
plotSPC(sp1, plot.order = rev(1:length(sp1)), width = 0.25)

# note that plotting order is derived from the call to `plotSPC(sp1)`
addBracket(b1, col='red', offset = -0.35)

# plot in reverse order
plotSPC(sp1, plot.order = rev(1:length(sp1)), width = 0.25)

# note that plotting order is derived from the call to `plotSPC(sp1)`
addBracket(b2, col='red', offset = -0.35)

```

addDiagnosticBracket *Annotate Diagnostic Features*

Description

Annotate diagnostic features within a sketch of soil profiles.

Usage

```
addDiagnosticBracket(  
  s,  
  kind,  
  feature = "featkind",  
  top = "featdept",  
  bottom = "featdepb",  
  ...  
)
```

Arguments

s	SoilProfileCollection object
kind	filter applied to feature column of diagnostic horizons registered within s
feature	column name containing feature kind
top	column name containing feature top depth
bottom	column name containing feature top depth
...	additional arguments passed to addBracket

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

[addBracket](#), [plotSPC](#)

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	integer, plotting character code
col	symbol color, either a single color or as many colors as there are horizons in x

Details

This function can only be called after plotting a SoilProfileCollection object. Details associated with a call to plotSPC are automatically accounted for within this function: e.g. plot.order, width, etc..

Author(s)

D.E. Beaudette

See Also

[plotSPC](#)

aggregateColor

Summarize Soil Colors

Description

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

Usage

```
aggregateColor(
  x,
  groups = "genhz",
  col = "soil_color",
  colorSpace = "CIE2000",
  k = NULL,
  profile_wt = 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")
colorSpace	the name of color space or color distance metric to use for conversion of aggregate colors to Munsell; either CIE2000 (color distance metric), LAB, or sRGB. Default = 'CIE2000'
k	single integer specifying the number of colors discretized via PAM (cluster package), see details
profile_wt	the name of a site-level attribute used to modify weighting, e.g. area

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$genhz <- generalize.hz(loafercreek$hname, 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 *Probabalistic Estimation of Soil Depth within Groups*

Description

Estimate the most-likely depth to contact within a collection of soil profiles. Consider `getSoilDepthClass` followed by group-wise percentile estimation as a faster alternative.

Usage

```

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

```

Arguments

<code>x</code>	a <code>SoilProfileCollection</code> object
<code>groups</code>	the name of a site-level attribute that defines groups of profiles within a collection
<code>crit.prob</code>	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
<code>name</code>	horizon-level attribute where horizon designation is stored, defaults to <code>hzdesgname(x)</code>
<code>p</code>	a REGEX pattern that matches non-soil genetic horizons
<code>...</code>	additional arguments to <code>slab</code>

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

# set horizon designation in SPC
hzdesgnname(sp1) <- 'name'

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

alignTransect

Calculate Relative Positions from Transect Data

Description

This function is used to support relative positioning of soil profiles by plotSPC, based on transect or gradient values typically associated with a site level attribute (e.g. elevation). Gradient values specified in `x` are translated to the range used by plotSPC (usually 1, length(SPC)) specified in `x.min` and `x.max`.

Usage

```
alignTransect(x, x.min, x.max, fix = TRUE, ...)
```

Arguments

<code>x</code>	numeric vector, describing values along a transect: distance, elevation, climatic variables, etc.. Typically sourced from the site level attributes of a SoilProfileCollection object. Order is not important.
<code>x.min</code>	numeric, lower boundary to relative position scale
<code>x.max</code>	numeric, upper boundary to relative position scale
<code>fix</code>	logical, attempt fixing overlapping positions with <code>fixOverlap</code>
<code>...</code>	additional arguments to <code>fixOverlap</code>

Value

list containing:

- `grad`: values of `x` in ascending order
- `order`: ordering vector of `x`
- `relative.pos`: elements of `x` translated to the new relative scale defined by `x.min` and `x.max`

Examples

```
data("sierraTransect")

# split transects
g <- subset(sierraTransect, transect == 'Granite')
a <- subset(sierraTransect, transect == 'Andesite')

g.p <- alignTransect(g$elev, x.min = 1, x.max = length(g), fix = FALSE)
a.p <- alignTransect(a$elev, x.min = 1, x.max = length(a), fix = FALSE)

op <- par(mar=c(2,0,0,2), mfrow=c(2,1))

plotSPC(g, width=0.25, name.style='center-center',
        cex.names=0.75,
        relative.pos = g.p$relative.pos, plot.order = g.p$order)

axis(1, at = g.p$relative.pos, labels = g.p$grad, line = -1.5)

plotSPC(a, width=0.25, name.style='center-center',
        cex.names=0.75,
        relative.pos = a.p$relative.pos, plot.order = a.p$order)

axis(1, at = a.p$relative.pos, labels = a.p$grad, line = -1.5)

par(op)
```

allocate

Allocate soil properties within various classification systems.

Description

Generic function to allocate soil properties to different classification schemes.

Usage

```
allocate(
  ...,
  to = c("FAO Salt Severity", "FAO Black Soil", "ST Diagnostic Features"),
```

```

    droplevels = TRUE
  )

```

Arguments

... arguments to specific allocation functions, see details and examples

to character specifying the classification scheme: FAO Salt Severity, FAO Black Soil (see details for the required ...)

droplevels logical indicating whether to drop unused levels in factors. This is useful when the results have a large number of unused classes, which can waste space in tables and figures.

Details

This function is intended to allocate a set of soil properties to an established soil classification scheme, such as Salt Severity or Black Soil. Allocation is semantically different from classification. While classification is the 'act' of developing a grouping scheme, allocation is the assignment or identification of measurements to a established class (Powell, 2008).

Usage Details:

Each classification scheme (to argument) uses a different set of arguments.

- FAO Salt Severity
 - **EC**: electrical conductivity column name, dS/m
 - **pH**: pH column name, saturated paste extract
 - **ESP**: exchangeable sodium percentage column name, percent
- FAO Black Soils
 - **object**: a `data.frame` or `SoilProfileCollection`
 - **pedonid**: pedon ID column name, required when `object` is a `data.frame`
 - **hztop**: horizon top depth column name, required when `object` is a `data.frame`
 - **hzbot**: horizon bottom depth column name, required when `object` is a `data.frame`
 - **OC**: organic carbon column name, percent
 - **m_chroma**: moist Munsell chroma column name
 - **m_value**: moist Munsell value column name
 - **d_value**: dry Munsell value column name
 - **CEC**: cation exchange capacity column name (NH₄OAc at pH 7), units of cmol(+)/kg soil
 - **BS**: base saturation column name (NH₄OAc at pH 7), percent
 - **tropical**: logical, data are associated with "tropical soils"
- ST Diagnostic Features
 - **object**: a `data.frame` or `SoilProfileCollection`
 - **pedonid**: pedon ID column name, required when `object` is a `data.frame`
 - **hzname**: horizon name column, required when `object` is a `data.frame`
 - **hztop**: horizon top depth column name, required when `object` is a `data.frame`
 - **hzbot**: horizon bottom depth column name, required when `object` is a `data.frame`
 - **texture**: soil texture class (USDA) column name

- **rupresblkcem**: rupture resistance column name
- **m_value**: moist Munsell value column name
- **m_chroma**: moist Munsell chroma column name
- **d_value**: dry Munsell value column name
- **BS**: base saturation column name (method ??), percent
- **OC**: organic carbon column name, percent
- **n_value**: ??
- **featkind**: ??

Value

A vector or data.frame object.

Note

The results returned by `allocate(to = "ST Diagnostic Features")` currently return a limited set of diagnostic features that are easily defined. Also, the logic implemented for some features does not include all the criteria defined in the Keys to Soil Taxonomy.

References

Abrol, I., Yadav, J. & Massoud, F. 1988. **Salt-affected soils and their management**. No. Bulletin 39. Rome, FAO Soils.

FAO. 2006. **Guidelines for soil description**. Rome, Food and Agriculture Organization of the United Nations.

FAO. 2020. DEFINITION | What is a black soil? (online). (Cited 28 December 2020). <http://www.fao.org/global-soil-partnership/intergovernmental-technical-panel-soils/gsoc17-implementation/internationalnetworkblacksoils/more-on-black-soils/definition-what-is-a-black-soil/es/>

Powell, B., 2008. Classifying soil and land, in: McKenzie, N.J., Grundy, M.J., Webster, R., Ringrose-Voase, A.J. (Eds.), **Guidelines for Survey Soil and Land Resources**, Australian Soil and Land Survey Handbook Series. CSIRO, Melbourne, p. 572.

Richards, L.A. 1954. **Diagnosis and Improvement of Saline and Alkali Soils**. U. S. Government Printing Office. 166 pp.

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

Examples

```
# Salt Severity
test <- expand.grid(
  EC = sort(sapply(c(0, 0.75, 2, 4, 8, 15, 30), function(x) x + c(0, -0.05, 0.05))),
  pH = c(8.1, 8.2, 8.3, 8.4, 8.5, 8.6),
  ESP = sort(sapply(c(0, 15, 30, 50, 70, 100), function(x) x + c(0, 0.1, -0.1)))
)
test$ss <- with(test, allocate(EC = EC, pH = pH, ESP = ESP, to = "FAO Salt Severity"))
table(test$ss)
```

```

# Black Soil Category 1 (BS1)
test <- expand.grid(
  dept = seq(0, 50, 10),
  OC = sort(sapply(c(0, 0.6, 1.2, 20, 40), function(x) x + c(0, -0.05, 0.05))),
  chroma_moist = 2:4,
  value_moist = 2:4,
  value_dry = 4:6,
  thickness = 24:26,
  CEC = 24:26,
  BS = 49:51,
  tropical = c(TRUE, FALSE)
)
test$pedon_id <- rep(1:21870, each = 6)
test$depb <- test$dept + 10

bs1 <- allocate(test, pedonid = "pedon_id", hztop = "dept", hzbot = "depb",
  OC = "OC", m_chroma = "chroma_moist", m_value = "value_moist",
  d_value = "value_dry", CEC = "CEC", BS = "BS",
  to = "FAO Black Soil"
)

table(BS1 = bs1$BS1, BS2 = bs1$BS2)

# SoilProfileCollection interface

data(sp3)
depths(sp3) <- id ~ top + bottom
hzdesgname(sp3) <- 'name'

# fake base saturation
horizons(sp3)$bs <- 75

plotSPC(sp3)

allocate(
  sp3,
  to = 'FAO Black Soil',
  OC = 'tc',
  m_chroma = 'chroma',
  m_value = 'value',
  d_value = 'value',
  CEC = 'cec',
  BS = 'bs'
)

# make a copy and edit horizon values
x <- sp3
x$value <- 2
x$chroma <- 2
x$cec <- 26
x$tc <- 2

```

```

x$soil_color <- munsell2rgb(x$hue, x$value, x$chroma)

plotSPC(x)

allocate(
  x,
  to = 'FAO Black Soil',
  OC = 'tc',
  m_chroma = 'chroma',
  m_value = 'value',
  d_value = 'value',
  CEC = 'cec',
  BS = 'bs'
)

# Soil Taxonomy Diagnostic Features
data(sp1)
df <- allocate(object = sp1, pedonid = "id", hzname = "name",
               hzdept = "top", hzdepb = "bot", texture = "texture",
               to = "ST Diagnostic Features"
)
aggregate(featdept ~ id, data = df, summary)

```

aqp_df_class

Get aqp_df_class entry from metadata or return a safe value.

Description

This is an accessor and replacement method for the aqp_df_class entry in the metadata slot. This entry is used internally by methods that interact with data.frame objects and slots to ensure that the same class used to promote to the SoilProfileCollection initially is used throughout the process.

Usage

```

## S4 method for signature 'SoilProfileCollection'
aqp_df_class(object)

## S4 replacement method for signature 'SoilProfileCollection'
aqp_df_class(object) <- value

```

Arguments

object	a SoilProfileCollection
value	"data.frame", "data.table" or "tbl_df"

`argillic.clay.increase.depth`*Return upper boundary of argillic horizon*

Description

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

Usage

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

Arguments

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

Details

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.

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, clay.attr = attr)
foo
```

as *Coerce SoilProfileCollection with as()*

Description

SoilProfileCollections can be coerced to other R object types using `as(spc, 'type')`.

Possible endpoints include: `list`, `data.frame`, `SpatialPointsDataFrame` and `SpatialPoints`.

Value

`list`
`data.frame`
`tbl_df`
`data.table`
`SpatialPointsDataFrame`
`SpatialPoints`

Examples

```
# load example data stored as SoilProfileCollection
data(sp5)

# sp5
str(sp5)

# list output
str(as(sp5, 'list'))

# data.frame output
str(as(sp5, 'data.frame'))

# Spatial Objects
# make some random coordinate data for each profile
sp5$x <- sp5$y <- rnorm(length(sp5))
coordinates(sp5) <- ~ x + y

# SpatialPointsDataFrame output
str(as(sp5, 'SpatialPointsDataFrame'))

# SpatialPoints output
str(as(sp5, 'SpatialPoints'))
```

barron.torrent.redness.LAB

Barron & Torrent (1986) Redness Index in LAB color space

Description

Calculate Redness Index after Barron & Torrent (1986) "Use of the Kubelka—Munk Theory to Study the Influence of Iron Oxides on Soil Colour" using Munsell colors converted to LAB. DOI: 10.1111/j.1365-2389.1986.tb00382.x. Accepts vectorized inputs for hue, value and chroma, produces vector output.

Usage

barron.torrent.redness.LAB(hue, value, chroma)

Arguments

hue	A character vector containing Munsell hues (e.g. "7.5YR")
value	A numeric vector containing Munsell values
chroma	A numeric vector containing Munsell chromas

Value

A numeric vector of horizon redness index (higher values = redder).

Author(s)

Andrew G. Brown.

References

Barron, V. and Torrent, J. (1986), Use of the Kubelka—Munk theory to study the influence of iron oxides on soil colour. *Journal of Soil Science*, 37: 499-510. doi:10.1111/j.1365-2389.1986.tb00382.x

bootstrapSoilTexture *Bootstrap Soil Texture Data*

Description

Simulate realistic sand/silt/clay values (a composition) using multivariate Normal distribution or Dirichlet distribution. Simulations from the multivariate Normal distribution are based on the compositional mean and variance-covariance matrix. Simulations from the Dirichlet distribution are based on maximum likelihood estimation of alpha parameters.

Usage

```
bootstrapSoilTexture(ssc, method = c("dirichlet", "normal"), n = 100)
```

Arguments

ssc	a data.frame object with 3 columns: sand, silt, clay and at least three rows of data within the range of 0-100 (percent). NA are automatically removed, but care should be taken to ensure that the sand/silt/clay values add to 100 percent. Simulations are based on these examples.
method	type of simulation: dirichlet or normal. See details.
n	number of simulated compositions. See details.

Details

Simulations from the multivariate normal distribution will more closely track the marginal distributions of sand, silt, and clay—possibly a better fit for "squished" compositions (TODO elaborate). However, these simulations can result in extreme (unlikely) estimates.

Simulations from the Dirichlet distribution will usually be a better fit (fewer extreme estimates) but require a fairly large number of records in ssc ($n \geq 30$?) for a reliable fit.

Additional examples will be added to [this tutorial](#).

Value

a list containing:

- samples - data.frame of simulated sand, silt, clay values
- mean - compositional mean
- var - compositional variance-covariance matrix
- D.alpha - (fitted) alpha parameters of the Dirichlet distribution, NULL when method = 'normal'

Note

This is a work in progress.

Author(s)

D.E. Beaudette

References

Aitchison, J. (1986) The Statistical Analysis of Compositional Data Monographs on Statistics and Applied Probability. Chapman & Hall Ltd., London (UK). 416p.

Aitchison, J, C. Barcel'o-Vidal, J.J. Egozcue, V. Pawlowsky-Glahn (2002) A concise guide to the algebraic geometric structure of the simplex, the sample space for compositional data analysis, Terra Nostra, Schriften der Alfred Wegener-Stiftung, 03/2003

Examples

```
if(
requireNamespace("compositions") &
  requireNamespace("soiltexture")
) {

  # sample data, data.frame
  data('sp4')

  # filter just Bt horizon data
  ssc <- sp4[grepl('^Bt', sp4$name), c('sand', 'silt', 'clay')]
  names(ssc) <- toupper(names(ssc))

  # simulate 100 samples
  s <- bootstrapSoilTexture(ssc, n = 100)
  s <- s$samples

  # empty soil texture triangle
  TT <- soiltexture::TT.plot(
    class.sys= "USDA-NCSS.TT",
    main= "",
    tri.sum.tst=FALSE,
    cex.lab=0.75,
    cex.axis=0.75,
    frame.bg.col='white',
    class.lab.col='black',
    lwd.axis=1.5,
    arrows.show=TRUE,
    new.mar = c(3, 0, 0, 0)
  )

  # add original data points
  soiltexture::TT.points(
    tri.data = s, geo = TT, col='firebrick',
    pch = 3, cex = 0.5, lwd = 1,
    tri.sum.tst = FALSE
  )

  # add simulated points
  soiltexture::TT.points(
    tri.data = ssc, geo = TT, bg='royalblue',
    pch = 22, cex = 1, lwd = 1,
    tri.sum.tst = FALSE
  )

  # simple legend
  legend('top',
        legend = c('Source', 'Simulated'),
        pch = c(22, 3),
        col = c('black', 'firebrick'),
```

```

    pt.bg = c('royalblue', NA),
    horiz = TRUE, bty = 'n'
  )
}

```

brierScore

Multinomial Brier Score

Description

Compute a multinomial Brier score from predicted class probabilities and observed class label. Lower values are associated with a more accurate classifier.

Usage

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

Arguments

x	data.frame of class probabilities (numeric) and observed class label (character), see examples
classLabels	vector of predicted class labels (probabilities), corresponding to column names in x
actual	name of column containing the observed class, should be character vector not factor

Value

a single Brier score, representative of data in x

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.

Examples

```

# columns 'a', 'b', 'c' contain predicted probabilities
# column 'actual' contains observed class label

# a good classifier
d.good <- data.frame(
  a = c(0.05, 0.05, 0.10),
  b = c(0.90, 0.85, 0.75),
  c = c(0.05, 0.10, 0.15),
  actual = c('b', 'b', 'b'),
  stringsAsFactors = FALSE
)

# a rather bad classifier
d.bad <- data.frame(
  a = c(0.05, 0.05, 0.10),
  b = c(0.90, 0.85, 0.75),
  c = c(0.05, 0.10, 0.15),
  actual = c('c', 'c', 'c'),
  stringsAsFactors = FALSE
)

# class labels are factors
d.factors <- data.frame(
  a = c(0.05, 0.05, 0.10),
  b = c(0.90, 0.85, 0.75),
  c = c(0.05, 0.10, 0.15),
  actual = c('b', 'b', 'b'),
  stringsAsFactors = TRUE
)

# relatively low value = accurate
brierScore(x = d.good, classLabels = c('a', 'b', 'c'), actual = 'actual')

# high values = not accurate
brierScore(x = d.bad, classLabels = c('a', 'b', 'c'), actual = 'actual')

# message related to conversion of factor -> character
brierScore(x = d.factors, classLabels = c('a', 'b', 'c'), actual = 'actual')

```

buntley.westin.index *Buntley-Westin (1965) Index*

Description

Calculate "Color Development Equivalent" by the method of Buntley & Westin (1965) "A Comparative Study of Developmental Color in a Chestnut-Chernozem-Brunizem Soil Climosequence" DOI: 10.2136/sssaj1965.03615995002900050029x. Originally developed for Mollisols, the Buntley-Westin index has been used as a tool to separate soils based on depth to particular colors.

Usage

```
buntley.westin.index(hue, chroma)
```

Arguments

hue	A character vector containing Munsell hues (e.g. "7.5YR")
chroma	A numeric vector containing Munsell chromas

Value

A numeric vector reflecting horizon color development.

Author(s)

Andrew G. Brown.

References

Buntley, G.J. and Westin, F.C. (1965), A Comparative Study of Developmental Color in a Chestnut-Chernozem-Brunizem Soil Climosequence. *Soil Science Society of America Journal*, 29: 579-582. doi:10.2136/sssaj1965.03615995002900050029x

c,SoilProfileCollection-method

Combine SoilProfileCollection objects

Description

Combine SoilProfileCollection objects or lists of SoilProfileCollection objects. This method provides ... expansion for the pbindlist method.

Usage

```
## S4 method for signature 'SoilProfileCollection'  
c(x, ...)  
  
## S4 method for signature 'SoilProfileCollection'  
combine(...)  
  
## S4 method for signature 'list'  
combine(...)
```

Arguments

x	A SoilProfileCollection
...	SoilProfileCollection objects

Value

A SoilProfileCollection

Examples

```
# example data
spc1 <- random_profile(1, SPC = TRUE)
spc2 <- random_profile(2, SPC = TRUE)
spc3 <- random_profile('A', SPC = TRUE)

# combine into a single SPC, ... interface
spc <- combine(spc1, spc2, spc3)

# combine into a single SPC, list interface
spc <- combine(list(spc1, spc2, spc3))

# input are combined into a single SPC
spc <- c(spc1, spc2, spc3)

# result is a list when a mixture of objects are provided
spc <- c(spc1, bar=spc2, baz="foo")
```

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 data.frame objects. 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://ncsslabdatamart.sc.egov.usda.gov/>

Examples

```
## Not run:
library(tactile)
library(lattice)
library(Hmisc)
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)

# 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,
  alpha=0.5,
  ylim=c(160,-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 = tactile.theme(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 = factor(mlra),
  data = a,
  lower=a$p.q25,
  upper=a$p.q75,
  alpha=0.25,
  sync.colors = TRUE,
  ylim=c(160,-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 = tactile.theme(superpose.line = list(lwd = 2)),
  auto.key = list(lines = TRUE, points = FALSE, columns = 2)
)

# 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'),
  par.settings = tactile.theme,
  layout = c(3,1)
)

# 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
par(mar = c(0, 0, 3, 1))
plotSPC(ca.1.to.10, name='hzn_desgn', color = 'CEC7')

## End(Not run)

```

Description

This function inspects a `SoilProfileCollection` object, looking for four common errors in horizon depths:

1. bottom depth shallower than top depth
2. equal top and bottom depth
3. missing top or bottom depth (e.g. NA)
4. gap or overlap between adjacent horizons

Usage

```
checkHzDepthLogic(
  x,
  hzdepths = NULL,
  idname = NULL,
  fast = FALSE,
  byhz = FALSE
)
```

Arguments

<code>x</code>	<code>SoilProfileCollection</code> or <code>data.frame</code> object to check
<code>hzdepths</code>	<code>SoilProfileCollection</code> uses <code>horizonDepths(x)</code> Default: <code>NULL</code> ; if <code>x</code> is a <code>data.frame</code> , character vector of column names of top and bottom depths
<code>idname</code>	<code>SoilProfileCollection</code> uses <code>idname(x)</code> Default: <code>NULL</code> ; if <code>x</code> is a <code>data.frame</code> , character vector with column name of unique profile ID;
<code>fast</code>	If details about specific test results are not needed, the operation can allocate less memory and run approximately 5x faster. Default: <code>FALSE</code>
<code>byhz</code>	Apply logic tests to profiles or individual horizons?

Value

A `data.frame` containing profile IDs, validity boolean (`valid`) and test results if `fast = FALSE`. The `data.frame` will have as many rows as profiles in `x` (`length(x)`).

- `id`: Profile IDs, named according to `idname(x)`
- `valid`: boolean, profile passes all of the following tests
 - `depthLogic`: boolean, errors related to depth logic
 - `sameDepth`: boolean, errors related to same top/bottom depths
 - `missingDepth`: boolean, NA in top / bottom depths
 - `overlapOrGap`: boolean, gaps or overlap in adjacent horizons

Author(s)

D.E. Beaudette, A.G. Brown, S.M. Roecker

Examples

```
## sample data

data(sp3)
depths(sp3) <- id ~ top + bottom

# these data should be clean
res <- checkHzDepthLogic(sp3)

head(res)

# less memory if only concerned about net validity
res <- checkHzDepthLogic(sp3, fast = TRUE)

head(res)
```

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. Use checkHzDepthLogic to check for common errors in horizon depths.

Value

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

Author(s)

D.E. Beaudette

See Also

[rebuildSPC](#), [checkHzDepthLogic](#)

`colorContrast`*Metrics of Contrast Suitable for Comparing Soil Colors*

Description

Pair-wise comparisons of Munsell color specifications, based on the NCSS color contrast classes and CIE delta-E 2000 metric.

Usage

```
colorContrast(m1, m2)
```

Arguments

<code>m1</code>	vector of Munsell colors ('10YR 3/3')
<code>m2</code>	vector of Munsell colors ('10YR 3/6')

Details

This function is fully vectorized but expects input to be of the same length. Use `expand.grid` to generate suitable input from 1:many or many:1 type comparisons. See [this tutorial](#) for an expanded discussion and more examples.

Value

A `data.frame` with the following columns:

- `m1`: Munsell color 1
- `m2`: Munsell color 2
- `dH`: delta-hue, as computed by `huePosition`
- `dV`: delta-value, absolute value of difference in Munsell value (`m1` vs. `m2`)
- `dc`: delta-chroma, absolute value of difference in Munsell chroma (`m1` vs. `m2`)
- `dE00`: delta-E00, e.g. the [CIE delta-E as refined in 2000](#)
- `cc`: soil color contrast class, as specified in [Soil Survey Technical Note 2](#)

Note

delta-E00 is computed by the [farver package](#).

Author(s)

D.E. Beaudette

References

1. https://en.wikipedia.org/wiki/Color_difference
2. [Soil Survey Technical Note 2](#)

See Also[colorContrastPlot](#), [huePosition](#)**Examples**

```
# two sets of colors to compare
m1 <- c('10YR 6/3', '7.5YR 3/3', '10YR 2/2', '7.5YR 3/4')
m2 <- c('5YR 3/4', '7.5YR 4/4', '2.5YR 2/2', '7.5YR 6/3')

# contrast metrics
colorContrast(m1, m2)

# adjacent chips
colorContrast('10YR 3/3', '10YR 3/4')
colorContrast('10YR 3/3', '7.5YR 3/3')

# highly contrasting colors
# http://colour.granjow.net/fabercastell-polychromos.html
colorContrastPlot('10B 4/13', '10YR 10/15',
  labels = c('helioblue-reddish', 'light cadmium yellow')
)
```

`colorContrastPlot`*Color Contrast Plot*

Description

A simple display of two sets of colors, NCSS color contrast class and CIE delta-E00.

Usage

```
colorContrastPlot(
  m1,
  m2,
  col.cex = 1,
  col.font = 2,
  d.cex = 1,
  cc.font = 3,
  dE00.font = 1,
  labels = c("m1", "m2"),
  label.cex = 1,
  label.font = 1,
  printMetrics = TRUE,
  ...
)
```

Arguments

m1	first set of Munsell colors for comparison (e.g. '5YR 3/2')
m2	second set of Munsell colors for comparison
col.cex	scaling factor for color labels
col.font	font for color labels
d.cex	contrast for contrast metric labels
cc.font	font for contrast class
dE00.font	font for delta-E00
labels	labels for compared colors, vector length 2
label.cex	scaling factor for labels
label.font	font for labels
printMetrics	logical, print metrics between color swatches
...	further arguments to <code>colorspace::swatchplot</code>

Details

This function requires the `farver` package for calculation of CIE delta-E00

Author(s)

D.E. Beaudette

See Also

[colorContrast](#)

Examples

```
# two sets of colors to compare
m1 <- c('10YR 6/3', '7.5YR 3/3', '10YR 2/2', '7.5YR 3/4')
m2 <- c('5YR 3/4', '7.5YR 4/4', '2.5YR 2/2', '7.5YR 6/3')

# contrast metrics
colorContrast(m1, m2)

# graphical display
colorContrastPlot(m1, m2)
```

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))
```

Arguments

soilColors vector of R colors (sRGB colorspace)
p marginal quantiles of interest

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 munsell) 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 Munsell chips, and dE00
- L1: L1 median CIELAB (D65) values, closest Munsell chip, and dE00

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)
```

compositeSPC

Return a list representation of site and horizon level data

Description

compositeSPC() is a convenience function that returns a named list representation of the columns from the @site and @horizons slots.

Usage

```
compositeSPC(object)
```

Arguments

object A SoilProfileCollection

Value

A list.

Author(s)

Andrew G. Brown.

confusionIndex	<i>Confusion Index</i>
----------------	------------------------

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)

# for comparison
shannonEntropy(p)
```

contrastChart *Color Contrast Chart*

Description

Compare one or more pages from a simulated Munsell book of soil colors to a reference color.

Usage

```
contrastChart(
  m,
  hues,
  ccAbbreviate = 1,
  style = "hue",
  gridLines = FALSE,
  thresh = NULL,
  returnData = FALSE
)
```

Arguments

m	Munsell representation of a single color for comparison e.g. '10YR 4/3'
hues	vector of one or more Munsell hue pages to display
ccAbbreviate	length of abbreviated contrast classes, use 0 to suppress labels
style	'hue' or 'CC', see details
gridLines	logical, add grid lines to the color contrast chart
thresh	threshold (<) applied to pair-wise comparisons and resulting color chips
returnData	logical, return lattice figure + data used to generate the figure

Details

A simulated Munsell color book page or pages are used to demonstrate color contrast between all chips and the reference color m (highlighted in red). NCSS color contrast class and CIE delta-E00 values are printed below all other color chips. Munsell color chips for chroma 5 and 7 are omitted, but axis labels are retained as a reminder of this fact.

Setting style='hue' emphasizes the contrast classes and CIE delta-E00 of chips adjacent to m. Setting style='CC' emphasizes adjacent chips according to respective contrast class via lattice panels.

Two-way panels are used when multiple hues are provided and style='CC'. The default output can be greatly enhanced via:

```
latticeExtra::useOuterStrips(..., strip = strip.custom(bg=grey(0.85)), strip.left =
strip.custom(bg=grey(0.85)) )
```

Author(s)

D.E. Beaudette

Examples

```
# single hue page
contrastChart(m = '10YR 3/3', hues = '10YR')

# multiple hue pages
contrastChart(m = '10YR 3/3', hues = c('10YR', '2.5Y'))

# contrast class, single hue
contrastChart(m = '10YR 3/3', hues = '10YR', style='CC')

# contrast class, multiple hues
# consider latticeExtra::useOuterStrips()
contrastChart(m = '10YR 5/6', hues = c('10YR', '2.5Y'), style='CC')
```

 contrastClass

Soil Color Contrast

Description

Determine soil color contrast class according to methods outlined in the Soil Survey Manual. This function is typically called from `colorContrast()` which is simpler to use and provides more information.

Usage

```
contrastClass(v1, c1, v2, c2, dH, dV, dC, verbose = FALSE)
```

Arguments

v1	Munsell value of first color
c1	Munsell chroma of first color
v2	Munsell value of second color
c2	Munsell chroma of second color
dH	delta Hue
dV	delta Value
dC	delta Chroma
verbose	return a list for testing rules/cases

Details

This function is fully vectorized but expects all inputs have the same length.

Value

A vector of color contrast classes (ordered factor). A list when `verbose` is TRUE.

Author(s)

D.E. Beaudette

References

- [Soil Survey Technical Note 2](#)

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

```
## standard use, result is an ordered factor
# 10YR 6/3 vs 5YR 3/4
contrastClass(v1=6, c1=3, v2=3, c2=4, dH=2, dV=3, dC=1)

## verbose output, useful for testing rules/cases
# 10YR 6/3 vs 5YR 3/4
contrastClass(v1=6, c1=3, v2=3, c2=4, dH=2, dV=3, dC=1, verbose = TRUE)
```

coordinates

Get coordinates from spatial slot

Description

Get coordinates from spatial slot, if present.

Usage

```
## S4 method for signature 'SoilProfileCollection'
coordinates(obj)

## S4 replacement method for signature 'SoilProfileCollection'
coordinates(object) <- value
```

Arguments

obj	a SoilProfileCollection
object	A SoilProfileCollection
value	A formula specifying columns containing x and y coordinates

Examples

```

data(sp5)

# coordinates are stored in x and y column of site
sp5$x <- rnorm(length(sp5))
sp5$y <- rnorm(length(sp5))

# coordinates takes a formula object as input
coordinates(sp5) <- ~ x + y

```

correctAWC

Apply rock fragment or salt correction to available water content

Description

Apply rock fragment or salt correction to available water content

Usage

```
correctAWC(awc, total_rf, gravel = NULL, ec = NULL)
```

Arguments

awc	Numeric vector of available water capacities (e.g. from estimateAWC)
total_rf	Numeric vector of rock fragment volume percentage, 0 - 100
gravel	Numeric vector of gravel volume percentage, 0 - 100
ec	Numeric vector of electrical conductivity, mmhos/cm

Value

A numeric vector (double) containing estimated available water capacities corrected for rock fragments and salts

Examples

```

# medium organic matter, loam texture
base.awc <- 0.18 # estimateAWC(texcl = "1", omcl = 2, na.rm = TRUE)

# medium organic matter, loam texture w/ 23% rock fragments by volume
corrected.awc <- correctAWC(base.awc, total_rf = 23)
corrected.awc

# medium organic matter, loam texture w/ 0% frags by volume and 8 mmhos/cm salts
salty.awc <- correctAWC(base.awc, total_rf = 0, ec = 8)
salty.awc

```

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.

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.

Details

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

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))
```

denormalize	<i>Create a (redundant) horizon-level attribute from a site-level attribute</i>
-------------	---

Description

Create a (redundant) horizon-level attribute from a site-level attribute. Specify a `SoilProfileCollection` and a site-level attribute from that SPC (by name) to receive a vector of length equal to the number of horizons containing the site-level values. This vector is directly usable with the `SoilProfileCollection` horizon setter.

`denormalize` is the inverse operation for the formula interface that "normalizes" a horizon level variable to site level:

```
site(object) <-~ horizonvar
```

Usage

```
denormalize(object, attr)
```

Arguments

<code>object</code>	A <code>SoilProfileCollection</code>
<code>attr</code>	Site-level attribute name (character string) to denormalize to horizon.

Details

"Denormalization" is the process of trying to improve the read performance of a database, at the expense of losing some write performance, by adding redundant copies of data or by grouping data. Sometimes it is beneficial to have site-level attributes denormalized for grouping of horizon-level data in analyses. `denormalize` achieves this result for `SoilProfileCollections`.

Value

A vector of values of equal length to the number of rows in the horizon table of the input SPC.

Author(s)

Andrew G. Brown, Dylan Beaudette

Examples

```

data(sp1)

# create a SoilProfileCollection from horizon data
depths(sp1) <- id ~ top + bottom

# create random site-level attribute `sitevar` with a binary (0/1) outcome
sp1$sitevar <- round(runif(length(sp1)))

# use denormalize() to create a mirror of sitevar in the horizon table
# name the attribute something different (e.g. `hz.sitevar`) to
# prevent collision with the site attribute
# the attributes can have the same name but you will then need
# site() or horizons() to access explicitly
sp1$hz.sitevar <- denormalize(sp1, 'sitevar')

# compare number of profiles to number of sitevar assignments
length(sp1)
table(sp1$sitevar)

# compare number of horizons to number of horizon-level copies of sitevar `hz.'sitevar`
nrow(sp1)
table(sp1$hz.sitevar)

```

depthOf

Get top or bottom depths of horizons matching a regular expression pattern

Description

The depthOf family of functions calculate depth of occurrence of a horizon designation pattern. They are used primarily in the place of complex qualitative or quantitative data that would confirm taxonomic criteria.

Generally, these functions are used to implement assumptions about relationships between diagnostic properties, features and materials and horizon designations commonly used in the field. Particular assumptions may not apply in all localities and/or data sources. Great care should be taken when inspecting results.

If you need all depths of occurrence for a particular pattern, depthOf is what you are looking for. minDepthOf and maxDepthOf are wrappers around depthOf that return the minimum and maximum depth. They are all set up to handle missing values and missing "contacts" with the target pattern.

Usage

```

depthOf(
  p,
  pattern,
  top = TRUE,

```

```

    hzdesgn = guessHzDesgnName(p),
    no.contact.depth = NULL,
    no.contact.assigned = NA
  )

```

Arguments

<code>p</code>	A single-profile <code>SoilProfileCollection</code> .
<code>pattern</code>	A regular expression to match in the horizon designation column. See: <code>hzdesgn</code>
<code>top</code>	Should the top (TRUE) or bottom (FALSE) depth be returned for matching horizons? Default: TRUE.
<code>hzdesgn</code>	Column name containing horizon designations. Default: <code>guessHzDesgnName(p)</code>
<code>no.contact.depth</code>	Depth to assume that contact did not occur.
<code>no.contact.assigned</code>	Depth to assign when a contact did not occur.

Value

A numeric vector containing specified depth(s) of horizons matching a pattern.

Author(s)

Andrew G. Brown

Examples

```

# construct a fake profile
spc <- data.frame(id=1, taxsubgrp = "Lithic Haploxerepts",
  hzname = c("A","AB","Bw","BC","R"),
  hzdept = c(0, 20, 32, 42, 49),
  hzdepb = c(20, 32, 42, 49, 200),
  clay = c(19, 22, 22, 21, NA),
  texcl = c("1","1","1", "1","br"),
  d_value = c(5, 5, 5, 6, NA),
  m_value = c(2.5, 3, 3, 4, NA),
  m_chroma = c(2, 3, 4, 4, NA))

# promote to SoilProfileCollection
depths(spc) <- id ~ hzdept + hzdepb
hzdesgnname(spc) <- 'hzname'
hztexclname(spc) <- 'texcl'

# multiple horizons contain B
depthOf(spc, "B")

# deepest top depth of horizon containing B
maxDepthOf(spc, "B")

# shallowest top depth

```

```

minDepthOf(spc, "B")

# deepest bottom depth
maxDepthOf(spc, "B", top = FALSE)

# deepest bottom depth above 35cm
maxDepthOf(spc, "B", top = FALSE, no.contact.depth = 35)

# assign infinity (Inf) if B horizon does not start within 10cm
minDepthOf(spc, "B", no.contact.depth = 10, no.contact.assigned = Inf)

```

```

depths<- ,SoilProfileCollection-method
Initialize a SoilProfileCollection from a data.frame object

```

Description

Initialize a SoilProfileCollection from a data.frame object

Usage

```

## S4 replacement method for signature 'SoilProfileCollection'
depths(object) <- value

## S4 replacement method for signature 'data.frame'
depths(object) <- value

```

Arguments

object	An object to promote to SoilProfileCollection (inherits from data.frame)
value	A formula specifying the unique profile ID, top and bottom depth column names

Examples

```

## init SoilProfileCollection objects from data.frame of horizon data

# load demo data
data(sp1)

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

# plot
plot(sp1)

# number of profiles
length(sp1)

```

```
# number of horizons
nrow(sp1)
```

depthWeights *Return a vector of contributing fractions over a depth interval*

Description

depthWeights() calculates the contributing fraction for each pair of horizon top and bottom depths, given an upper and lower boundary.

Usage

```
depthWeights(top, bottom, upper, lower)
```

Arguments

top	A numeric vector of horizon top depths.
bottom	A numeric vector of horizon bottom depths.
upper	A unit length numeric vector with upper boundary.
lower	A unit length numeric vector with lower boundary.

Value

A named list.

Author(s)

Andrew G. Brown.

depth_units *Get depth units from metadata*

Description

Get units of depth measurement from metadata. Default value is centimeters.

Arguments

object	A SoilProfileCollection
value	character, a value representing units. Default 'cm'.

Examples

```

data(sp5)

## get depth units
du <- depth_units(sp5)

# set alternate units; e.g. inches
depth_units(sp5) <- 'in'

# replace original value (cm)
depth_units(sp5) <- du

```

diagnostic_hz	<i>Retrieve diagnostic data from SoilProfileCollection</i>
---------------	--

Description

Get diagnostic data from SoilProfileCollection. Result is returned in the same `data.frame` class used to initially construct the SoilProfileCollection.

Arguments

object	a SoilProfileCollection
--------	-------------------------

diagnostic_hz<-	<i>Add data to the diagnostic slot</i>
-----------------	--

Description

Diagnostic data in an object inheriting from `data.frame` can easily be added via `merge` (LEFT JOIN). There must be one or more same-named columns containing profile ID on the left and right hand side to facilitate the join: `diagnostic_hz(spc) <-newdata`

Arguments

object	A SoilProfileCollection
value	An object inheriting <code>data.frame</code>

Examples

```

# load test data
data(sp2)

# promote to SPC
depths(sp2) <- id ~ top + bottom

# assign two profiles a zone related to the mollic epipedon
newdata <- data.frame(id = c("hon-1", "hon-17"),
                      featkind = "fixed-depth surface sample",
                      featdept = 0,
                      featdepb = 18)

# do left join
diagnostic_hz(sp2) <- newdata

# inspect site table: newvalue TRUE only for horizons
# with top depth equal to zero
diagnostic_hz(sp2)

```

dice

*Efficient Slicing of SoilProfileCollection Objects***Description**

Cut ("dice") soil horizons into 1-unit thick slices. This function replaces `slice`.

Usage

```

dice(
  x,
  fm = NULL,
  SPC = TRUE,
  pctMissing = FALSE,
  fill = FALSE,
  strict = TRUE,
  byhz = FALSE
)

```

Arguments

<code>x</code>	a <code>SoilProfileCollection</code> object
<code>fm</code>	optional formula describing top depths and horizon level attributes to include: <code>integer.vector ~ var1 + var2 + var3</code> or <code>integer.vector ~ .</code> to include all horizon level attributes. When <code>NULL</code> profiles are "diced" to depth and results will include all horizon level attributes.

SPC	return the diced SoilProfileCollection, if FALSE a data.frame of horizon-level attributes
pctMissing	compute "percent missing data" by slice (when TRUE expect 6-8x longer run time)
fill	logical, fill with empty placeholder horizons in gaps within profiles, and/or, above/below interval specified in fm. Automatically set to TRUE when fm is specified. Backwards compatibility with slice is maintained by setting fill = TRUE with or without fm.
strict	perform horizon depth logic checking / flagging / removal
byhz	Evaluate horizon depth logic at the horizon level (TRUE) or profile level (FALSE). Invalid depth logic invokes HzDepthLogicSubset which removes offending profiles or horizon records.

Value

a SoilProfileCollection object, or data.frame when SPC = FALSE

Author(s)

D.E. Beaudette, A.G. Brown

duplicate

Duplicate Profiles of a SoilProfileCollection

Description

A simple function to duplicate the contents of a SoilProfileCollection object. Old profile IDs are saved as a site-level attribute (oldID) and new IDs are generated using a numeric serial number.

Usage

```
duplicate(x, times = 3, oldID = ".oldID")
```

Arguments

x	a SoilProfileCollection object with 1 or more profiles
times	requested number of copies
oldID	site-level attribute used to store the original profile IDs

Value

a SoilProfileCollection object

Author(s)

D.E. Beaudette

Examples

```
# sample data
data('sp4')

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

# duplicate each profile 2 times
d <- duplicate(sp4, times = 2)

# graphical check
par(mar = c(0, 0, 3, 1))
plotSPC(d, color = 'Ca', width = 0.25)
```

equivalentMunsellChips

Identify "equivalent" (whole number value/chroma) Munsell chips

Description

Uses a pre-calculated lookup list ([equivalent_munsell](#)) based on pair-wise CIE2000 contrast (dE00) of LAB color with D65 illuminant for all whole value/chroma "chips" in the `aqp:munsell` data set.

The intention is to identify Munsell chips that may be "functionally equivalent" to some other given whole value/chroma chip elsewhere in the Munsell color space – as discretized in the `aqp:munsell` data table. This basic assumption needs to be validated against your end goal: probably by visual inspection of some or all of the resulting sets. See [colorContrast](#) and [colorContrastPlot](#).

"Equivalent" chips table are based (fairly arbitrarily) on the 0.001 probability level of dE00 (default Type 7 quantile) within the upper triangle of the 8467x8467 contrast matrix. This corresponds to a dE00 contrast threshold of approximately 2.15.

Usage

```
equivalentMunsellChips(hue = NULL, value = NULL, chroma = NULL)
```

Arguments

hue	A character vector containing Munsell hues
value	A numeric vector containing Munsell values (integer only)
chroma	A numeric vector containing Munsell chromas (integer only)

Value

A named list; Each list element contains a data.frame with one or more rows of "equivalent" Munsell, RGB and LAB color coordinates from `munsell` data set.

References

- Gaurav Sharma, Wencheng Wu, Edul N. Dalal. (2005). The CIEDE2000 Color-Difference Formula: Implementation Notes, Supplementary Test Data, and Mathematical Observations. *COLOR research and application*. 30(1):21-30. <http://www2.ece.rochester.edu/~gsharma/ciede2000/ciede2000noteCRNA.pdf>
- Thomas Lin Pedersen, Berendea Nicolae and Romain François (2020). farver: High Performance Colour Space Manipulation. R package version 2.0.3. <https://CRAN.R-project.org/package=farver>
- Dong, C.E., Webb, J.B., Bottrell, M.C., Saginor, I., Lee, B.D. and Stern, L.A. (2020). Strengths, Limitations, and Recommendations for Instrumental Color Measurement in Forensic Soil Characterization. *J Forensic Sci*, 65: 438-449. <https://doi.org/10.1111/1556-4029.14193>

See Also

[colorContrast](#) [colorContrastPlot](#) [equivalent_munsell](#)

Examples

```
# 7.5YR 4/4 (the one and only)

equivalentMunsellChips("7.5YR", 4, 4)
#>
#> `$`7.5YR 4/4`
#>      hue value chroma      r      g      b      L      A      B
#> 8330 7.5YR      4      4 0.4923909 0.352334 0.2313328 41.26403 10.8689 23.5914

# 7.5YR 1/1 (two chips are equivalent; 3 row result)

equivalentMunsellChips("7.5YR", 1, 1)
#>
#> `$`7.5YR 1/1`
#>      hue value chroma      r      g      b      L      A      B
#> 1983 10YR      1      1 0.1345633 0.1087014 0.07606787 10.64787 1.621323 6.847629
#> 6189  5YR      1      1 0.1330994 0.1076359 0.09450179 10.63901 2.489012 3.515146
#> 8303 7.5YR      1      1 0.1329483 0.1082380 0.08862581 10.64210 2.065514 4.623922

# 10YR 6/8 (two chips are equivalent; 3 row result)

equivalentMunsellChips("10YR", 6, 8)
#>
#> `$`10YR 6/8`
#>      hue value chroma      r      g      b      L      A      B
#> 2039 10YR      6      7 0.7382230 0.5512957 0.2680260 61.76795 10.50886 44.78574
#> 2040 10YR      6      8 0.7519872 0.5472116 0.2157209 61.77496 11.83215 51.15496
#> 2041 10YR      6      9 0.7642826 0.5433189 0.1559069 61.78085 13.09599 57.49773

# compare visually a very red color

veryred <- equivalentMunsellChips("10R", 6, 28)[[1]]

par(mar=c(0,0,1,1))
```

```

pie(rep(1, nrow(veryred)), col = with(veryred, munsell2rgb(hue, value, chroma)),
    label = with(veryred, sprintf("%s %s/%s", hue, value, chroma)))

table(veryred$hue) # 2 hues
#>
#> 10R 7.5R
#> 8 17

table(veryred$value) # 2 values
#>
#> 5 6
#> 11 14

table(veryred$chroma) # 10 chromas
#>
#> 21 22 23 24 25 26 27 28 29 30
#> 1 2 2 3 3 4 3 3 2 2

```

equivalent_munsell *Indices of "equivalent" Munsell chips in munsell data set*

Description

A pre-calculated lookup list (made with `farver::compare_colour`) based on pair-wise CIE2000 color contrast (dE_{00}) of LAB colors with D65 illuminant for all whole value/chroma "chips" in the `aqp:munsell` data set.

The intention is to identify Munsell chips that may be "functionally equivalent" to some other given whole chip elsewhere in the Munsell color space – as discretized in the `aqp:munsell` lookup table.

"Equivalent" chips are based (fairly arbitrarily) on the 0.001 probability level of dE_{00} (default Type 7 quantile) within the upper triangle of the 8467×8467 contrast matrix. This corresponds to a dE_{00} contrast threshold of approximately 2.15.

This is a naive (to the subtleties of human color perception, and overall magnitude of contrast between some of the "chips") but computationally consistent approach. Using the lookup list, as opposed to manual contrast via e.g. `farver::compare_colour` may have some benefits for efficiency in certain applications where the exact contrast value is not as important as the concept of having some threshold that is non-zero, but very small.

Usage

```
data(equivalent_munsell)
```

Format

A named list with 8467 elements, each containing a numeric vector of indices corresponding to the munsell data set, which has 8467 rows (unique, whole-number chips). Names have the format HUE VALUE/CHROMA, eg. "7.5YR 4/4"

References

- Gaurav Sharma, Wencheng Wu, Edul N. Dalal. (2005). The CIEDE2000 Color-Difference Formula: Implementation Notes, Supplementary Test Data, and Mathematical Observations. *COLOR research and application*. 30(1):21-30. <http://www2.ece.rochester.edu/~gsharma/ciede2000/ciede2000noteCRNA.pdf>
- Thomas Lin Pedersen, Berendea Nicolae and Romain Francois (2020). *farver*: High Performance Colour Space Manipulation. R package version 2.0.3. <https://CRAN.R-project.org/package=farver>
- Dong, C.E., Webb, J.B., Bottrell, M.C., Saginor, I., Lee, B.D. and Stern, L.A. (2020). Strengths, Limitations, and Recommendations for Instrumental Color Measurement in Forensic Soil Characterization. *J Forensic Sci*, 65: 438-449. <https://doi.org/10.1111/1556-4029.14193>

See Also

[equivalentMunsellChips](#)

Examples

```
data(equivalent_munsell)
```

estimateAWC

Estimate available water capacity for fine-earth fraction

Description

Estimate available water capacity for fine-earth fraction

Usage

```
estimateAWC(texcl, omcl, precision = 2, FUN = mean, ...)
```

Arguments

texcl	character, USDA textural class fine earth fraction
omcl	integer, Organic matter class. 1: less than 1.5 percent, 2: less than 5, 3: greater than 5
precision	integer, Number of decimal places in result default: 2
FUN	Function for interpolating between table values default: mean
...	Additional arguments to FUN

Value

A numeric vector double containing estimated available water capacities for fine-earth fraction.

Examples

```
# organic matter, loam texture, low medium and high OM
base.awc <- estimateAWC(c("1","1","1"), c(1, 2, 3), na.rm = TRUE)
base.awc
```

estimatePSCS	<i>Estimate boundaries of the particle size control section (U.S Soil Taxonomy; 12th edition)</i>
--------------	---

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).

Usage

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

Arguments

p	A single-profile SoilProfileCollection object
hzdesgn	Name of the horizon attribute containing the horizon designation. Default 'hzname'
clay.attr	Name of the horizon attribute containing clay contents. Default 'clay'
texcl.attr	Name of the horizon attribute containing textural class (used for finding sandy textures). Default 'texcl'
tax_order_field	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.
bottom.pattern	Regular expression pattern to match a root-restrictive contact. Default matches Cr, R or Cd. This argument is passed to both estimateSoilDepth and getArgillicBounds.
...	additional arguments are passed to getArgillicBounds()

Details

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.

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', clay.attr = attr, texcl.attr="texture")
foo
```

estimateSoilDepth	<i>Estimate Soil Depth</i>
-------------------	----------------------------

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",  
  p = "Cr|R|Cd",  
  selection = min,  
  no.contact.depth = NULL,  
  no.contact.assigned = NULL  
)
```

Arguments

f	SoilProfileCollection object of length 1, e.g. a single profile
name	name of the column that contains horizon designations
p	REGEX pattern for determining "contact", or depth to some morphologic feature (e.g. Bt)
selection	function applied in the presence of multiple matching horizons: min (default), max, mean, etc.
no.contact.depth	in the absence of contact matching p, a depth at which we can assume a standard depth-to-contact
no.contact.assigned	value assigned when no contact is encountered at or below no.contact.depth

Details

The choice of a selection function usually follows:

min: the top of the first matching horizon, max: the top bot the last matching horizon, or possibly mean: somewhere in-between.

Value

single value representing the depth to contact or no.contact.assigned

Author(s)

D.E. Beaudette and J.M. Skovlin

See Also

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

Examples

```
## consider a situation where there were multiple candidate
## "contacts": 2 Cd horizons over an R

# init hypothetical profile
d <- data.frame(
  id = '1',
  top = c(0, 10, 20, 30, 40, 50, 60),
  bottom = c(10, 20, 30, 40, 50, 60, 80),
  name = c('A', 'Bt1', 'Bt2', 'BC', 'Cd1', 'Cd2', 'R'),
  stringsAsFactors = FALSE
)

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

# visual check
par(mar = c(0, 0, 0, 1))
plotSPC(d, hz.depths = TRUE, name.style = 'center-center', cex.names = 1)

# top of the first Cd
estimateSoilDepth(d, name = 'name')

# top of the first Cd
estimateSoilDepth(d, name = 'name', selection = min)

# top of the R
estimateSoilDepth(d, name = 'name', selection = max)

# top of the second Cd
estimateSoilDepth(d, name = 'name', selection = max, p = 'Cd')

## another example

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')

# 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',
    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
plotSPC(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", method = "relative")
```

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
method	'relative' (proportion of total) or 'absolute' depths

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.

Value

A vector values ranging from 0 to 1 (method='relative') or 0 to maximum depth in specified depth units (method='absolute'), representing the quantity 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$data.complete <- evalMissingData(sp2, vars = c('r', 'g', 'b'), name = 'name')
sp2$data.complete.abs <- evalMissingData(sp2, vars = c('r', 'g', 'b'),
```

```

name = 'name', method = 'absolute')

# rank
new.order <- order(sp2$data.complete)

# plot along data completeness ranking
plot(sp2, plot.order=new.order, name='name')

# add relative completeness axis
# note re-ordering of axis labels
axis(side=1, at=1:length(sp2), labels = round(sp2$data.complete[new.order], 2),
      line=-1.5, cex.axis=0.75)

# add absolute completeness (cm)
axis(side=1, at=1:length(sp2), labels = sp2$data.complete.abs[new.order],
      line=1, cex.axis=0.75)

```

explainPlotSPC

Visual Explanation for plotSPC

Description

Create a visual explanation for the many arguments to `plotSPC`. Call this function instead of `plotSPC`, all objects after `x` are passed on to `plotSPC`. Nearly all of the figures in the [Introduction to SoilProfileCollection Objects tutorial](#) are created with this function.

Usage

```
explainPlotSPC(x, ...)
```

Arguments

`x` a `SoilProfileCollection` object
`...` arguments passed to [plotSPC](#)

Value

a list of internally-used ordering vectors and graphical offsets / scaling factors

Author(s)

D.E. Beaudette

See Also

[plotSPC](#)

Examples

```

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

# proposed vector of relative positions, overlap likely
pos <- c(1, 1.1, 3, 4, 5, 5.2, 7, 8, 9, 10)

# try it
explainPlotSPC(sp4, name='name', relative.pos=pos)

# attempt to fix using an integer sequence, short-circuit will prevent adjustments
explainPlotSPC(sp4, name='name', relative.pos=fixOverlap(1:10))

# attempt to adjust using defaults
explainPlotSPC(sp4, name='name', relative.pos=fixOverlap(pos))

# attempt to adjust and tinker with defaults
explainPlotSPC(sp4, name='name', relative.pos=fixOverlap(pos, adj = 0.2))

# repeatable adjustments
set.seed(10101)
explainPlotSPC(sp4, name='name', relative.pos=fixOverlap(pos, thresh = 0.7))

# more complex adjustments required
pos <- c(1, 2, 3, 3.3, 5, 5.1, 5.5, 8, 9, 9.1)

# tinker
explainPlotSPC(sp4, name='name', relative.pos=pos)
explainPlotSPC(sp4, name='name', relative.pos=fixOverlap(pos))
explainPlotSPC(sp4, name='name', relative.pos=fixOverlap(pos, thresh = 0.7))
explainPlotSPC(sp4, name='name', relative.pos=fixOverlap(pos, thresh=0.7, adj = 0.2))

# no solution possible given these constraints
explainPlotSPC(sp4, name='name', relative.pos=fixOverlap(pos, thresh=1, adj = 0.2))

```

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.

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 + clinochlore
+ 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(l), 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)
#

```

fillHzGaps

Find and Fill Horizon Gaps

Description

This function attempts to find "gaps" in the horizon records of a `SoilProfileCollection` object and fill with placeholder horizons (profile ID, horizon ID, to/bottom depths, all else NA). Missing

horizon records between the top of each profile and `to_top`, or the bottom of each profile and `to_bottom` are treated as gaps when those arguments are not `NULL`. You can use this function to prepare a potentially messy `SoilProfileCollection` for subsequent analyses that are sensitive to horizon sequence inconsistencies or require a conformal "rectangle" of data spanning known depths.

Gaps are defined as:

- within each profile, for horizons `i` to `n_hz`:
- `bottom_i != top_{i+1}` (but only to `i = 1:(n_hz - 1)`)

Usage

```
fillHzGaps(x, flag = TRUE, to_top = 0, to_bottom = max(x))
```

Arguments

<code>x</code>	SoilProfileCollection object
<code>flag</code>	logical, flag empty horizons that have been added. default: <code>TRUE</code>
<code>to_top</code>	numeric, fill from shallowest top depth in each profile to specified depth? default: <code>0</code>
<code>to_bottom</code>	numeric, fill from deepest bottom depth in each profile to specified depth? default: <code>aqp::max(x)</code>

Value

SoilProfileCollection object

Author(s)

A.G. Brown and D.E. Beaudette

Examples

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

# introduce depth logic errors
idx <- c(2, 6:7, 8, 12)
sp4$top[idx] <- NA

# check
horizons(sp4)[idx, ]

# create gaps by removing logic errors
x <- HzDepthLogicSubset(sp4, byhz = TRUE)

# inspect
par(mar = c(0, 0, 0, 1))
plotSPC(x, width = 0.3, default.color = 'royalblue', name = 'hzID')
```

```

z <- fillHzGaps(x, flag = TRUE)

plotSPC(z, width = 0.3, color = '.filledGap', name = 'hzID', show.legend = FALSE)

# fill top to 0 cm
z2 <- fillHzGaps(x, flag = TRUE, to_top = 0)
plotSPC(z2, width = 0.3, color = '.filledGap', name = 'hzID', show.legend = FALSE)

# fill bottom to max(SPC)
z3 <- fillHzGaps(x, flag = TRUE, to_top = 0, to_bottom = max(x))
plotSPC(z3, width = 0.3, color = '.filledGap', name = 'hzID', show.legend = FALSE)

## another example
data(sp4)
depths(sp4) <- id ~ top + bottom

# remove 1st horizons from profiles 1:4
idx <- sp4[, , .FIRST, .HZID]
replaceHorizons(sp4) <- horizons(sp4)[-idx[1:4], ]

# prepare for dice()
z <- fillHzGaps(sp4, to_top = 0, to_bottom = 50, flag = TRUE)

# empty-horizon padding is in place for formula interface to dice()
d <- dice(z, fm = 0:50 ~ .)
plotSPC(d, color = 'Ca', show.legend = FALSE)
plotSPC(d, color = '.filledGap', show.legend = FALSE)

```

findOverlap

Find Overlap within a Sequence

Description

Establish which elements within a vector of horizontal positions overlap beyond a given threshold

Usage

```
findOverlap(x, thresh)
```

Arguments

x	vector of relative horizontal positions, one for each profile
thresh	threshold defining "overlap", typically < 1

Value

unique index to affected (overlapping) elements in x

Examples

```
x <- c(1, 2, 3, 3.4, 3.5, 5, 6, 10)

findOverlap(x, thresh = 0.5)
```

fixOverlap

Fix Overlap within a Sequence via Simulated Annealing

Description

This function makes small adjustments to elements of `x` until overlap defined by `thresh` is removed, or until `maxIter` is reached. Rank order and boundary conditions (defined by `min.x` and `max.x`) are preserved. The underlying algorithm is based on simulated annealing. The "cooling schedule" parameters `T0` and `k` can be used to tune the algorithm for specific applications.

Usage

```
fixOverlap(
  x,
  thresh = 0.6,
  adj = thresh * 2/3,
  min.x = min(x) - 0.2,
  max.x = max(x) + 0.2,
  maxIter = 1000,
  trace = FALSE,
  tiny = 1e-04,
  T0 = 500,
  k = 10
)
```

Arguments

<code>x</code>	vector of horizontal positions
<code>thresh</code>	horizontal threshold defining "overlap" or distance between elements of <code>x</code> . For adjusting soil profile sketches values are typically < 1 and likely in (0.3, 0.8).
<code>adj</code>	specifies the size of perturbations within <code>runif(min = adj * -1, max = adj)</code> . Larger values will sometimes reduce the number of iterations required to solve particularly difficult overlap conditions. See <code>coolingRate</code> argument when <code>adj</code> is large
<code>min.x</code>	left-side boundary condition, consider expanding if a solution cannot be found within <code>maxIter</code> .
<code>max.x</code>	right-side boundary condition, consider expanding if a solution cannot be found within <code>maxIter</code> .
<code>maxIter</code>	maximum number of iterations to attempt before giving up and returning a regularly-spaced sequence

trace	print diagnostics, result is a list vs vector
tiny	the smallest allowable overlap
T0	starting temperature
k	cooling constant

Details

Ideas for solving difficult overlap scenarios:

- widen the boundary conditions by adjusting `min.x` and `max.x` beyond the original scale of `x`
- reduce the allowable overlap threshold `thresh`
- reduce the magnitude of perturbations (`adj`) and increase `maxIter`
- increase `k`

Value

When `trace = FALSE`, a vector of the same length as `x`, preserving rank-ordering and boundary conditions. When `trace = TRUE` a list containing the new sequence along with information about objective functions and decisions made during iteration.

Author(s)

D.E. Beaudette

Examples

```
x <- c(1, 2, 3, 3.4, 3.5, 5, 6, 10)

# easy
z <- fixOverlap(x, thresh = 0.2, trace = TRUE)

# harder
z <- fixOverlap(x, thresh = 0.6, trace = TRUE)

# much harder
z <- fixOverlap(x, thresh = 0.9, trace = TRUE)

# interpret `trace` output

# relatively challenging
x <- c(1, 2, 3.4, 3.4, 3.4, 3.4, 6, 8, 10, 12, 13, 13, 15, 15.5)

# fix overlap, return debugging information
set.seed(10101)
z <- fixOverlap(x, thresh = 0.8, trace = TRUE)

# setup plot device
par(mar = c(4, 4, 1, 1))
```

```

layout(matrix(c(1,2,3)), widths = 1, heights = c(1,1,2))

# objective function = overlap + SSD
plot(
  seq_along(z$stats), z$stats,
  type = 'h', las = 1,
  xlab = 'Iteration', ylab = 'Overlap',
  cex.axis = 0.8
)

# SSD: deviation from original configuration
plot(
  seq_along(z$ssd), z$ssd,
  type = 'h', las = 1,
  xlab = 'Iteration', ylab = 'Deviation',
  cex.axis = 0.8
)

# adjustments at each iteration
matplot(
  z$states, type = 'l',
  lty = 1, las = 1,
  xlab = 'Iteration', ylab = 'x-position'
)

# trace log
# B: boundary condition violation
# O: rank (order) violation
# +: accepted perturbation
# -: rejected perturbation
table(z$log)

```

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 = "not-used", hzdepn = NA, ...)
```

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
...                   additional arguments passed to grep() such as perl=TRUE for advanced REGEX

```

Value

factor of the same length as x

Author(s)

D.E. Beaudette

Examples

```

## Not run:

data(sp1)

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

# generalize
sp1$genhz <- generalize.hz(sp1$name,
                          new=c('0', 'A', 'B', 'C', 'R'),
                          pat=c('0', '^A', '^B', 'C', 'R'))

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

## a more advanced example, requires perl=TRUE
# example data
x <- c('A', 'AC', 'Bt1', '^AC', 'C', 'BC', 'CB')

# new labels
n <- c('A', '^AC', 'C')
# patterns:
# "A anywhere in the name"
# "literal '^A' anywhere in the name"
# "C anywhere in name, but without preceding A"
p <- c('A', '^A', '(?!A)C')

# note additional argument
res <- generalize.hz(x, new = n, pat=p, perl=TRUE)

# double-check: OK
table(res, x)

## End(Not run)

```

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.

Usage

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

Arguments

p	a SoilProfileCollection, containing a single profile
attr	horizon attribute name to get the "increase" of
threshold.fun	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

Details

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

Value

Returns a numeric vector of depths where the increase requirement is met. For the argillic, the first is the one of interest.

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.depths(p, threshold.fun = crit.clay.argillic,
                           attr = attr, vertical.distance = 30)

foo

```

get.increase.matrix	<i>compute pair-wise distances to determine where an attribute increases within a vertical distance specified</i>
---------------------	---

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.

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 <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

Details

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.

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 = attr(x, which = "original.levels"))
```

Arguments

x	output from slab
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: https://en.wikipedia.org/wiki/Brier_score . 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	<i>Estimate upper and lower boundary of argillic diagnostic subsurface horizon</i>
-------------------	--

Description

getArgillicBounds estimates the 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. The function 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 small, and in which horizon.

Usage

```
getArgillicBounds(
  p,
  hzdesgn = "hzname",
  clay.attr = "clay",
  texcl.attr = "texcl",
  require_t = TRUE,
  bottom.pattern = "Cr|R|Cd",
  lower.grad.pattern = "[2-9]*B*CB*[^rtd]*[1-9]*$",
  sandy.texture.pattern = "-S$|^S$|COS$|L[^V]FS$|[^L]VFS$|LS$|LFS$",
  verbose = FALSE
)
```

Arguments

p	A single-profile SoilProfileCollection
hzdesgn	the name of the column/attribute containing the horizon designation; default="hzname"
clay.attr	the name of the column/attribute containing the clay content; default="clay"
texcl.attr	the name of the column/attribute containing the textural class (used for finding sandy horizons); default="texcl"
require_t	require a "t" subscript for positive identification of upper and lower bound of argillic? default: TRUE
bottom.pattern	regular expression passed to estimateSoilDepth to match the lower boundary of the soil. default is "Cr RICd" 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.
sandy.texture.pattern	this is a pattern for matching sandy textural classes: -S\$ ^S\$ COS\$ L[^V]FS\$ [^L]VFS\$ LS\$ LFS\$

verbose Print out information about 't' subscripts, sandy textures, plow layers and lower gradational horizons?

Details

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]*B*CB*[^rtd]*[1-9]*$`. 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.

The minimum thickness of the argillic horizon is dependent on whether all subhorizons are "sandy" or not. The `sandy.texture.pattern` default `-S$|^S$|COS$L[^V]FS$[^L]VFS$|LS$|LFS$` captures USDA textural class fine earth fractions that meet "sandy" particle size class criteria.

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.

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 G. Brown

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', clay.attr = attr, texcl.attr="texture")
foo
```

getCambicBounds *Find all intervals that are potentially part of a Cambic horizon*

Description

Find all intervals that are potentially part of a Cambic horizon excluding those that are part of an argillic horizon (defined either by depth interval or getArgillicBounds()).

There may be multiple cambic horizons (indexes) in a profile. Each cambic index has a top and bottom depth associated: cambic_top and cambic_bottom. This result is designed to be used for single profiles, or with profileApply(..., frameify = TRUE)

Usage

```
getCambicBounds(
  p,
  hzdesgn = guessHzDesgnName(p),
  texcl.attr = guessHzTexClName(p),
  clay.attr = guessHzAttrName(p, attr = "clay", c("total", "_r")),
  argi_bounds = NULL,
  d_value = "d_value",
  m_value = "m_value",
  m_chroma = "m_chroma",
  sandy.texture.pattern = "-S$|^S$|COS$|L[^V]FS$|[^L]VFS$|LS$|LFSS$",
  ...
)
```

Arguments

p	A single-profile SoilProfileCollection
hzdesgn	Column name containing horizon designations.
texcl.attr	Arguments to getArgillicBounds()
clay.attr	Arguments to getArgillicBounds()
argi_bounds	Optional: numeric vector of length 2 with top and bottom of argillic; (Default: NULL)
d_value	Column name containing dry value. Default: d_value
m_value	Column name containing moist value. Default: m_value
m_chroma	Column name containing moist chroma Default: m_chroma
sandy.texture.pattern	this is a pattern for matching sandy textural classes: -S\$ ^S\$ COS\$ L[^V]FS\$ [^L]VFS\$ LS\$ LFSS\$
...	Arguments to getArgillicBounds()

Value

A data.frame containing profile, cambic indexes, along with top and bottom depths.

Author(s)

Andrew G. Brown

Examples

```
# construct a fake profile
spc <- data.frame(id=1, taxsubgrp = "Lithic Haploxerepts",
  hzname = c("A", "AB", "Bw", "BC", "R"),
  hzdept = c(0, 20, 32, 42, 49),
  hzdepb = c(20, 32, 42, 49, 200),
  clay = c(19, 22, 22, 21, NA),
  texcl = c("1", "1", "1", "1", "br"),
  d_value = c(5, 5, 5, 6, NA),
  m_value = c(2.5, 3, 3, 4, NA),
  m_chroma = c(2, 3, 4, 4, NA))

# promote to SoilProfileCollection
depths(spc) <- id ~ hzdept + hzdepb
hzdesgname(spc) <- 'hzname'
hztexclname(spc) <- 'texcl'

# print results in table
getCambicBounds(spc)
```

getClosestMunsellChip *Get Approximate Munsell Chip*

Description

Non-standard Munsell notation (e.g. '7.9YR 2.7/2.0') can be matched (nearest-neighbor, no interpolation) to the closest color within the `munsell` sRGB/CIELAB look-up table via `getClosestMunsellChip()`. A more accurate estimate of sRGB values from non-standard notation can be achieved with the [munsellinterpol](#) package.

Usage

```
getClosestMunsellChip(munsellColor, convertColors = TRUE, ...)
```

Arguments

`munsellColor` character vector of strings containing Munsell notation of color, e.g. '10YR 4/3'
`convertColors` logical, should parsed Munsell colors be converted into sRGB values
`...` further arguments to `munsell2rgb`

Value

a `data.frame` when `convertColors=TRUE`, otherwise character vector

Examples

```
# 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')
```

getLastHorizonID	<i>Get IDs of Deepest Horizons by Profile</i>
------------------	---

Description

Return horizon IDs of the deepest horizon within each profile of a `SoilProfileCollection`. IDs are returned in the same order as `profile_id(x)`. Horizon top depths are used because there are cases where bottom depths may be missing.

Usage

```
getLastHorizonID(x)
```

Arguments

x a `SoilProfileCollection`

getSoilDepthClass	<i>Generate Soil Depth Class Matrix</i>
-------------------	---

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')

# 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. For instance: horizon designations: A1-A2-A3-C-Ab , would return A3 bottom depth given pattern = "A".

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.

Usage

```
getMineralSoilSurfaceDepth(p, hzdesgn = guessHzDesgnName(p), pattern = "O")
getPlowLayerDepth(p, hzdesgn = guessHzDesgnName(p), pattern = "^Ap[^b]*")
```

Arguments

p	A single-profile SoilProfileCollection object.
pattern	Regular expression pattern to match for all horizons to be considered part of the "surface".
hzdesgn	Column name containing horizon designation. Default: guessHzDesgnName(p).

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 G. Brown

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,SoilProfileCollection-method

Subset soil horizon data using a depth or depth interval

Description

glom() returns a "clod" of horizons from a SoilProfileCollection from a depth interval.

All horizons included within the specified interval are returned in their entirety (not just the portion within the interval), unless the truncate argument is specified. Horizon intersection is based on unique ID hzidname(spc) and attribute of interest.

If intersection at the specified boundaries [`'z1'`, `'z2'`] results in no horizon data, 'NULL' is returned with a warning containing the offending pedon ID.

If inverting results with invert, it is possible that thick horizons (that span more than the entire glom interval) will be split into two horizons. This may make the results from `ids = TRUE` different from what you expect, as they will be based on a profile with an "extra" horizon.

Usage

```

## S4 method for signature 'SoilProfileCollection'
glom(
  p,
  z1,
  z2 = NULL,
  ids = FALSE,
  df = FALSE,
  truncate = FALSE,
  invert = FALSE,
  modality = "all"
)

```

Arguments

p A SoilProfileCollection

z1	Top depth (required) - depth to intersect horizon; if 'z2' specified, top depth of intersection interval.
z2	optional: bottom depth of intersection interval
ids	Return only horizon IDs? default: FALSE
df	Return a data.frame, by intersection with horizons(p)? default: FALSE
truncate	Truncate horizon top and bottom depths to z1 and z2? default: FALSE
invert	Get the horizons ranges outside the interval z1/z2? default: FALSE
modality	Return all horizons (default: "all") or first, <i>thickest</i> (modality = "thickest") horizon in interval.

Details

The verb/function that creates a clod is "glom". "To glom" is "to steal" or to "become stuck or attached to". The word is related to the compound "glomalin", which is a glycoprotein produced by mycorrhizal fungi in soil.

Value

A SoilProfileCollection, data.frame, or a vector of horizon IDs. NULL if no result.

Author(s)

Andrew G. Brown

See Also

[glomApply trunc](#)

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)
```

glomApply

*Subset an SPC by applying glom to each profile***Description**

glomApply() is a function used for subsetting SoilProfileCollection objects by depth. It is a wrapper around glom which is intended to subset single-profile SPCs based on depth intervals/intersection.

glomApply works by accepting a function .fun as argument. This function is used on each profile to process a multi-profile SPC for input to glom (via profileApply). For each profile, .fun returns a 2-length numeric vector of top and bottom boundaries glom arguments: z1, z2.

glomApply provides the option to generate profile-specific glom depths for a large SPC and handles iteration and rebuilding of a subset SPC object. Optional arguments include: truncate to cut the boundaries to specified [z1, z2]; invert to the portion outside [z1, z2], modality to either "all" horizons or "thickest" horizon in the glom interval. ... are various expressions you can run on the individual profiles using NSE, similar to mutate.

Usage

```
glomApply(
  object,
  .fun = NULL,
  truncate = FALSE,
  invert = FALSE,
  modality = "all",
  ...,
  chunk.size = 100
)
```

Arguments

object	A SoilProfileCollection
.fun	A function that returns vector with top and bottom depth (z1 and z2 arguments to glom) for a single profile p (as passed by profileApply)
truncate	Truncate horizon top and bottom depths to [z1, z2]
invert	Truncate horizon top and bottom depths to [z1, z2] and then invert result?
modality	Aggregation method for glom result. Default "all": return all horizons; "thickest": return (shallowest) thickest horizon
...	A set of comma-delimited R expressions that resolve to a transformation to be applied to a single profile e.g glomApply(hzdept = max(hzdept) - hzdept) like aqp::mutate
chunk.size	Chunk size parameter for profileApply

Value

A SoilProfileCollection.

Author(s)

Andrew G. Brown.

See Also

[glom trunc](#)
[glom glomApply](#)

Examples

```
data(sp3)
depths(sp3) <- id ~ top + bottom

# constant depths, whole horizon returns by default
plot(glomApply(sp3, function(p) c(25,100)))

# constant depths, truncated
#(see aqp::trunc for helper function)
plot(glomApply(sp3, function(p) c(25,30), truncate = TRUE))

# constant depths, inverted
plot(glomApply(sp3, function(p) c(25,100), invert = TRUE))

# constant depths, inverted + truncated (same as above)
plot(glomApply(sp3, function(p) c(25,30), invert = TRUE, truncate=TRUE))

# random boundaries in each profile
plot(glomApply(sp3, function(p) round(sort(runif(2, 0, max(sp3))))))

# random boundaries in each profile (truncated)
plot(glomApply(sp3, function(p) round(sort(runif(2, 0, max(sp3))))), truncate = TRUE))

# calculate some boundaries as site level attributes
sp3$glom_top <- profileApply(sp3, getMineralSoilSurfaceDepth)
sp3$glom_bottom <- profileApply(sp3, estimateSoilDepth)

# use site level attributes for glom intervals for each profile
plot(glomApply(sp3, function(p) return(c(p$glom_top, p$glom_bottom))))
```

grepSPC

Subset SPC with pattern-matching for text-based attributes

Description

grepSPC() is a shorthand function for subsetting SoilProfileCollections. For example, by `filter(grepl(spc, ...))` or `filter(stringr::str_detect(spc, ...))`. It provides pattern matching for a single text-based site or horizon level attribute.

Arguments

object	A SoilProfileCollection
attr	A character vector (column in object) for matching patterns against.
pattern	REGEX pattern to match in attr
...	Additional arguments are passed to grep()

Value

A SoilProfileCollection.

Author(s)

Andrew G. Brown.

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 left-right ordering of groups can be adjusted by converting groups into a factor and explicitly setting factor levels. Alpha-numeric ordering is used for all other types.

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

# add a groups
sp1$group.2 <- sprintf("%s-%s", rev(LETTERS[1:3]), sp1$group)

# convert fake group to factor with new levels
sp1$group.3 <- factor(sp1$group.2, levels=c('C-2', 'B-2', 'A-2', 'C-1', 'B-1', 'A-1'))

# plot profiles, sorted and annotated by 'group' (integers)
par(mar=c(1,1,1,1))
groupedProfilePlot(sp1, groups='group', max.depth=150, group.name.offset = -5, id.style='side')

# plot profiles, sorted and annotated by 'group.2' (characters)
par(mar=c(1,1,1,1))
groupedProfilePlot(sp1, groups='group.2', max.depth=150, group.name.offset = -5, id.style='side')

# plot profiles, sorted and annotated by 'group.3' (characters)
par(mar=c(1,1,1,1))
```

```

groupedProfilePlot(sp1, groups='group.3', max.depth=150, group.name.offset = -5, 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='hillslopeprof', group.name.cex = 0.5,
group.name.offset = -10)

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

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

## End(Not run)

```

groupSPC	<i>(EXPERIMENTAL) Store groupings within a profile collection.</i>
----------	--

Description

(EXPERIMENTAL) Store groupings within a profile collection.

Usage

```
groupSPC(object, ...)
```

Arguments

object	SoilProfileCollection.
...	One or more expressions evaluated within the context of object that resolve to vectors that can be coerced to factor "groups."

guessGenHzLevels	<i>Guess Appropriate Ordering for Generalized Horizon Labels</i>
------------------	--

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)
```

guessHzAttrName	<i>Guess Arbitrary Horizon Column Name</i>
-----------------	--

Description

Guess the horizon column name where possible/preferred formative elements are known. There is a preference for records where more optional requirements are met to handle cases where there will be many matches. For example, working with soil data one might have "low, RV and high" total clay, as well as clay fractions. One could distinguish between these different measurements using standard formative elements for column names from the database of interest. Result is the first match in `horizonNames(x)` with the most required plus optional patterns matched.

e.g. `guessHzAttrName(x, attr="clay", optional=c("total", "_r"))` matches `(claytotal_r == totalclay_r)` over `(clay_r == claytotal == totalclay)` over `clay`.

Usage

```
guessHzAttrName(x, attr, optional = NULL, verbose = TRUE)
```

Arguments

<code>x</code>	A <code>SoilProfileCollection</code>
<code>attr</code>	A regular expression containing required formative element of attribute name.
<code>optional</code>	A character vector of regular expression(s) containing optional formative elements of attribute name.
<code>verbose</code>	A boolean value for whether to produce message output about guesses.

Value

Character containing horizon attribute column name. Result is the first match in `horizonNames(x)` with the most required plus optional patterns matched.

Author(s)

Andrew G. Brown

See Also

[guessHzDesgnName](#), [guessHzTexClName](#)

Examples

```
# a has the required attr pattern, but none of the optional
a <- data.frame(id = 1, top = c(0,10), bottom=c(10,40),
               clay=c(18,19))
depths(a) <- id ~ top + bottom
```

```

guessHzAttrName(a, attr="clay", optional=c("total", "_r"))

# b has required attr pattern, and one of the optional patterns
# notice that it also contains "clay" but preferentially matches more optional patterns
b <- data.frame(id = 1, top = c(0,10), bottom=c(10,40),
               clay=c(0.18,0.19), clay_r=c(18,19))
depths(b) <- id ~ top + bottom

guessHzAttrName(b, attr="clay", optional=c("total", "_r"))

# c has total and _r (both optional) on either side of clay
# having all of the optional patterns plus required is best evidence, and first
# column containing that combination will be returned
c <- data.frame(id = 1, top = c(0,10), bottom=c(10,40),
               totalclay_r=c(18,19), claytotal_r=c(0.18,0.19))
depths(c) <- id ~ top + bottom

guessHzAttrName(c, attr="clay", optional=c("total", "_r"))

```

guessHzDesignName	<i>Guess Horizon Designation Column Name</i>
-------------------	--

Description

This follows the historic convention used by `aqp::plotSPC()` looking for "hzname" or other column names containing the regular expression "name". If the pattern "name" is not found, the pattern "desgn" is searched as a fallback, as "hzdesgn" or "hz_desgn" are other common column naming schemes for horizon designation name.

Usage

```
guessHzDesignName(x)
```

Arguments

x A `SoilProfileCollection`

Value

Character containing horizon designation column name.

Author(s)

Andrew G. Brown

See Also

[guessHzTexC1Name](#), [guessHzAttrName](#)

Examples

```
a <- data.frame(id = 1, top = c(0,10), bottom=c(10,40), horizonname=c("A","Bw"))
depths(a) <- id ~ top + bottom

# store guess in metadata
hzdesgnname(a) <- guessHzDesgnName(a)

# inspect result
hzdesgnname(a)
```

guessHzTexClName	<i>Guess Horizon Texture Class Column Name</i>
------------------	--

Description

This function is used to provide a texture class attribute column name to functions. It will use regular expressions to match "texcl" which is typically the texture of the fine earth fraction, without modifiers or in-lieu textures. Alternately, it will match "texture" for cases where "texcl" is absent (e.g. in NASIS Component Horizon).

Usage

```
guessHzTexClName(x)
```

Arguments

x A SoilProfileCollection

Value

Character containing horizon texture class column name.

Author(s)

Andrew G. Brown

See Also

[guessHzDesgnName](#), [guessHzAttrName](#)

Examples

```
a <- data.frame(id = 1, top = c(0,10), bottom=c(10,40), texture=c("A","Bw"))
depths(a) <- id ~ top + bottom

# store guess in metadata
hzdesgnname(a) <- guessHzTexClName(a)

# inspect result
hzdesgnname(a)
```

harden.melanization *Harden (1982) Melanization*

Description

Calculate Melanization component of Profile Development Index after Harden (1982) "A quantitative index of soil development from field descriptions: Examples from a chronosequence in central California". Accepts vectorized inputs for value and reference value to produce vector output. A convenient use case would be to apply this on a profile-specific basis, where the `value_ref` has a single value, and `value` is a vector of length equal to the number of horizons within the upper 100 cm.

Usage

```
harden.melanization(value, value_ref)
```

Arguments

<code>value</code>	numeric vector containing Munsell values
<code>value_ref</code>	A numeric vector containing Munsell value(s) for reference material

Details

In Harden (1982), melanization is calculated relative to a reference parent material for all horizons within 100cm of the soil surface. In addition, several other non-color components are normalized relative to a maximum value and summed to obtain the overall Profile Development Index.

Value

A numeric vector reflecting horizon darkening relative to a reference (e.g. parent) material.

Author(s)

Andrew G. Brown.

References

Harden, J.W. (1982) A quantitative index of soil development from field descriptions: Examples from a chronosequence in central California. *Geoderma*. 28(1) 1-28. doi: 10.1016/0016-7061(82)90037-4

Examples

```
library(aqp)
data("jacobs2000", package="aqp")

# LEFT JOIN hue, value, chroma matrix color columns
horizons(jacobs2000) <- cbind(horizons(jacobs2000)[,c(idname(jacobs2000), hzidname(jacobs2000))],
                             parseMunsell(jacobs2000$matrix_color_munsell, convertColors = FALSE))

# calculate a mixed 150-200cm color ~"parent material"

jacobs2000$c_horizon_color <- profileApply(jacobs2000, function(p) {

  # and derive the parent material from the 150-200cm interval
  p150_200 <- glom(p, 150, 200, truncate = TRUE)
  p150_200$thickness <- p150_200$bottom - p150_200$top

  # mix colors
  clr <- na.omit(horizons(p150_200)[,c('matrix_color_munsell','thickness')])
  mixMunsell(clr$matrix_color_munsell, w = clr$thickness)$munsell

})

# segment profile into 1cm slices (for proper depth weighting)
jacobs2000$melan <- profileApply(jacobs2000, function(p) {

  # sum the melanization index over the 0-100cm interval
  p0_100 <- segment(p, 0:100)

  ccol <- parseMunsell(p$c_horizon_color, convertColors = FALSE)

  sum(harden.melanization(
    value = as.numeric(p0_100$value),
    value_ref = as.numeric(ccol$value)), na.rm = TRUE)

})

jacobs2000$melanorder <- order(jacobs2000$melan)

# Plot in order of increasing Melanization index

plotSPC(jacobs2000, axis.line.offset = -1,
        color = "matrix_color",
        label = "melan",
        plot.order = jacobs2000$melanorder)

abline(h = c(0,100,150,200), lty = 2)
```

```
# Add [estimated] parent material color swatches
lapply(seq_along(jacobs2000$c_horizon_color), function(i) {
  rect(i - 0.15, 250, i + 0.15, 225,
      col = parseMunsell(jacobs2000$c_horizon_color[jacobs2000$melanorder[i]]))
})
```

harden.rubification *Harden (1982) Rubification*

Description

Calculate Rubification component of Profile Development Index after Harden (1982) "A quantitative index of soil development from field descriptions: Examples from a chronosequence in central California". Accepts vectorized inputs for hue and chroma to produce vector output.

In Harden (1982) rubification is calculated relative to a reference parent material. Several other non-color components are normalized relative to a maximum value and summed to obtain the overall Profile Development Index.

Usage

```
harden.rubification(hue, chroma, hue_ref, chroma_ref)
```

Arguments

hue	A character vector containing Munsell hues (e.g. "7.5YR")
chroma	A numeric vector containing Munsell chromas
hue_ref	A character vector containing Munsell hue(s) (e.g. "10YR") for reference material
chroma_ref	A numeric vector containing Munsell chroma(s) for reference material

Value

A numeric vector reflecting horizon redness increase relative to a reference (e.g. parent) material.

Author(s)

Andrew G. Brown

References

Harden, J.W. (1982) A quantitative index of soil development from field descriptions: Examples from a chronosequence in central California. *Geoderma*. 28(1) 1-28. doi: 10.1016/0016-7061(82)90037-4

Examples

```

library(aqp)
data("jacobs2000", package="aqp")

# LEFT JOIN hue, value, chroma matrix color columns
horizons(jacobs2000) <- cbind(horizons(jacobs2000)[,c(idname(jacobs2000), hzidname(jacobs2000))],
                             parseMunsell(jacobs2000$matrix_color_munsell, convertColors = FALSE))

#' # calculate a mixed 150-200cm color ~"parent material"
jacobs2000$c_horizon_color <- profileApply(jacobs2000, function(p) {

  # and derive the parent material from the 150-200cm interval
  p150_200 <- glom(p, 150, 200, truncate = TRUE)
  p150_200$thickness <- p150_200$bottom - p150_200$top
  # mix colors
  clr <- na.omit(horizons(p150_200)[,c('matrix_color_munsell', 'thickness')])
  mixMunsell(clr$matrix_color_munsell, w = clr$thickness)$munsell
})

# segment profile into 1cm slices (for proper depth weighting)
jacobs2000$rubif <- profileApply(jacobs2000, function(p) {

  # sum the melanization index over the 0-100cm interval
  p0_100 <- segment(p, 0:100)

  ccol <- parseMunsell(p$c_horizon_color, convertColors = FALSE)

  sum(harden.rubification(
    hue = p0_100$hue,
    chroma = as.numeric(p0_100$chroma),
    hue_ref = ccol$hue,
    chroma_ref = as.numeric(ccol$chroma)
  ), na.rm = TRUE)

})

jacobs2000$rubiforder <- order(jacobs2000$rubif)

# Plot in order of increasing Rubification index

plotsPC(jacobs2000, axis.line.offset = -1,
        color = "matrix_color",
        label = "rubif",
        plot.order = jacobs2000$rubiforder)

abline(h = c(0,100,150,200), lty = 2)

# Add [estimated] parent material color swatches
lapply(seq_along(jacobs2000$c_horizon_color), function(i) {
  rect(i - 0.15, 250, i + 0.15, 225,
      col = parseMunsell(jacobs2000$c_horizon_color[jacobs2000$rubiforder[i]]))
})

```

harmonize,SoilProfileCollection-method

Harmonize a property by profile-level denormalization for convenient visualization or analysis of ranges

Description

It is sometimes convenient to be able to "denormalize" to a SoilProfileCollection with fewer attributes but more profiles. This is helpful wherever calculations are made on a profile basis and ranges or repeated measures are depicted with multiple attributes per soil horizon.

harmonize is most commonly used for creating "comparison" soil profile sketches with plotSPC—where the thematic attribute is derived from multiple data sources or summary statistics (such as quantiles of a property for Low-RV-High). However, the method more generally applies wherever one wants to alias between multiple columns containing "similar" data as input to an algorithm.

Data are "harmonized" to a common attribute names specified by the names of list elements in `x.names`. Profiles are essentially duplicated. In order to satisfy uniqueness constraints of the SoilProfileCollection, the label from the sub-elements of `x.names` are used to disambiguate profiles. A new column in the site table is calculated to reflect these groupings and facilitate filtering. See examples below.

Usage

```
## S4 method for signature 'SoilProfileCollection'
harmonize(x, x.names, keep.cols = NULL, grp.name = "hgroup")
```

Arguments

<code>x</code>	A SoilProfileCollection.
<code>x.names</code>	a named list of character vectors specifying target names, profile ID suffixes and source attribute names for harmonization
<code>keep.cols</code>	a character vector of column names to keep unaltered from the horizon data
<code>grp.name</code>	a character vector with column name to store grouping variable in site table (default: "hgroup")

Details

If attributes reflecting the same or similar property within a soil layer have different names (e.g. socQ05, socQ50, socQ95) it is sometimes inconvenient to work with them as multiple attributes within the same profile. These similar attributes may need to be analyzed together, or in sequence by profile, displayed using the same name or using a common scale. It is also useful to be able to alias different data sources that have the same attributes with different names.

Each list element in `x.names` specifies a single "harmonization," which is comprised of one or more mappings from new to old. Each named "sub-element" of `x.names` specifies the name and attribute to use for updating the profile ID and site table of the duplicated profiles.

Value

A (redundant) SoilProfileCollection, with one profile for each set of harmonizations specified by x.names.

Author(s)

Andrew G. Brown

Examples

```
### single source "harmonization" of single-profile with range -> single attribute, multi-profile

# make some test data
spc <- pbindlist(lapply(1:10, random_profile, SPC = TRUE))

# assume that p1, p2 and p3 are the low RV and high quantiles for a hypothetical property "foo"
h1 <- harmonize(spc, x.names = list(foo = c(q05 = "p1", q50 = "p2", q95 = "p3")))

# inspect result
plot(h1, color = "foo")

# filter with calculated "harmonized group" to get just RV profiles
plot(subset(h1, hgroup == "q50"), color="foo")

### single source, two properties at once; with common labels: "method1" "method2"

# assume that p1, p2 are measurements by two (!=) methods for a hypothetical property "foo"
#           p3, p4 are measurements by same two methods for a hypothetical property "bar"
h3 <- harmonize(spc, x.names = list(foo = c(method1 = "p1", method2 = "p2"),
                                     bar = c(method1 = "p3", method2 = "p4")))

plot(h3, color = "foo")
plot(h3, color = "bar")
head(horizons(h3))

# a slight modification, "method 1" only used for "foo" and "method 3" for "bar"
h3 <- harmonize(spc, x.names = list(foo = c(method1 = "p1", method2 = "p2"),
                                     bar = c(method2 = "p3", method3 = "p4")))

plot(h3, color = "foo") # note the pattern of values missing for foo (*_method3)
plot(h3, color = "bar") # likewise for bar (*_method1)

#' the new labels need not match across harmonizations -- not sure how useful this is but it works
h3 <- harmonize(spc, x.names = list(foo = c(method1 = "p1", method2 = "p2"),
                                     bar = c(method3 = "p3", method4 = "p4")))

plot(h3, color = "foo") # note the pattern of values missing for foo (*_method 3 + 4)
plot(h3, color = "bar") # likewise for bar (*_method 1 + 2)

### two-source harmonization

# make test data
spc1 <- pbindlist(lapply(LETTERS[1:5], random_profile, SPC = TRUE))
spc2 <- pbindlist(lapply(letters[1:5], random_profile, SPC = TRUE))
```

```

h4 <- pbindlist(list(harmonize(spc1, list(foo = c(transect1 = "p4"))), # foo is p4 in dataset 1
                  harmonize(spc2, list(foo = c(transect2 = "p2"))))) # foo is p2 in dataset 2

# same property with different name in two different datasets
plot(h4, color = "foo")

### many source harmonization

# make test datasets (n=10); highly redundant IDs (1:3 repeated)
spcs <- lapply(1:10, function(x) pbindlist(lapply(1:3, random_profile, SPC = TRUE)))

# randomly varying column name for demo (in each dataset, foo could be p1 thru p5)
rcolname <- paste0("p", round(runif(10, 0.5, 5.5)))

# iterate over data sources
bigspc <- pbindlist(lapply(1:length(spcs), function(i) {

  # assume each data source has a unique name for the property "foo"
  xn <- rcolname[i]

  # set names attribute to be equal to index i [creating unique profile IDs]
  # i.e. 2_10 will be profile ID 2 from 10th dataset
  names(xn) <- i

  # harmonize each data source, using unique column name and target name "foo"
  harmonize(spcs[[i]], x.names = list(foo = xn))
}))

# inspect a subset
plot(bigspc[1:30,], color = "foo")

```

hasDarkColors	<i>Find horizons with colors darker than a Munsell hue, value, chroma threshold</i>
---------------	---

Description

hasDarkColors returns a boolean value by horizon representing whether darkness thresholds are met. The code is fully vectorized and deals with missing data and optional thresholds.

Default arguments are set up for "5-3-3 colors" – the basic criteria for Mollic/Umbric epipedon/mineral soil darkness. Any of the thresholds or column names can be altered. Any thresholds that are set equal to NA will be ignored.

Usage

```

hasDarkColors(
  p,

```

```

d_hue = NA,
m_hue = NA,
d_value = 5,
d_chroma = NA,
m_value = 3,
m_chroma = 3,
dhuenm = "d_hue",
dvalnm = "d_value",
dchrnm = "d_chroma",
mhuenm = "m_hue",
mvalnm = "m_value",
mchrnm = "m_chroma"
)

```

Arguments

p	A SoilProfileCollection.
d_hue	Optional: character vector of dry hues to match (default: NA)
m_hue	Optional: character vector of moist hues to match (default: NA)
d_value	Maximum value of dry value (default: 5)
d_chroma	Optional: Maximum value of dry chroma (default: NA)
m_value	Maximum value of moist value (default: 3)
m_chroma	Maximum value of moist chroma (default: 3)
dhuenm	Column name containing dry hue.
dvalnm	Column name containing dry value.
dchrnm	Column name containing dry chroma.
mhuenm	Column name containing moist hue.
mvalnm	Column name containing moist value.
mchrnm	Column name containing moist chroma.

Value

Boolean value (for each horizon in p) reflecting whether "darkness" criteria are met.

Author(s)

Andrew G. Brown

Examples

```

# construct a fake profile
spc <- data.frame(id=1, taxsubgrp = "Lithic Haploxeralfs",
  hzdesgn = c("A","AB","Bt","BCt","R"),
  hzdept = c(0, 20, 32, 42, 49),
  hzdepb = c(20, 32, 42, 49, 200),
  d_value = c(5, 5, 5, 6, NA),

```

```

      m_value = c(2.5, 3, 3, 4, NA),
      m_chroma = c(2, 3, 4, 4, NA))

# promote to SoilProfileCollection
depths(spc) <- id ~ hzdept + hzdepb

# print results in table
data.frame(id = spc[[idname(spc)]],
           hz_desgn = spc$hzdesgn,
           has_dark_colors = hasDarkColors(spc))

```

horizonColorIndices *Horizon Color Indices*

Description

Calculate basic horizon-level color indices for a SoilProfileCollection. Basic indices do not require aggregation over the whole profile or comparison to a "reference" (e.g. parent material) color. Includes Hurst (1977) Redness Index, Barron-Torrent Redness Index (1986) and Buntley-Westin Index (1965). This is a wrapper method around several horizon-level indices. See the individual functions for more details.

Usage

```
horizonColorIndices(p, hue = "m_hue", value = "m_value", chroma = "m_chroma")
```

Arguments

p	A SoilProfileCollection
hue	Column name containing moist hue; default: "m_hue"
value	Column name containing moist value; default: "m_value"
chroma	Column name containing moist chroma; default: "m_chroma"

Value

A data.frame containing unique pedon and horizon IDs and horizon-level color indices.

Author(s)

Andrew G. Brown.

See Also

[hurst.redness.barron.torrent.redness.LAB.buntley.westin.index](#)

Examples

```

data(sp1)

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

# move site data
site(sp1) <- ~ group

# compute indices
# merged into `sp1` with left-join on hzidname(sp1)
horizons(sp1) <- horizonColorIndices(sp1, hue="hue", value="value", chroma="chroma")

# visualize
par(mar=c(0, 1, 3, 1))
plot(sp1, color='hurst_redness')
plot(sp1, color='barron_torrent_redness')
plot(sp1, color='buntley_westin')

```

horizonDepths<- *Set horizon depth column names*

Description

Set column name containing horizon ID

Get column names containing horizon depths

Arguments

value	a character vector of length two with names of columns containing numeric top and bottom depths
object	a SoilProfileCollection

horizonNames<- *Set horizon column names*

Description

Set horizon column names

Get names of columns in horizon table.

Arguments

value	a unique vector of equal length to number of columns in horizons <code>length(horizonNames(object))</code>
object	a SoilProfileCollection

horizons	<i>Retrieve horizon data from SoilProfileCollection</i>
----------	---

Description

Get horizon data from SoilProfileCollection. Result is returned in the same `data.frame` class used to initially construct the SoilProfileCollection.

Horizon data in an object inheriting from `data.frame` can easily be added via `merge (LEFT JOIN)`. There must be one or more same-named columns (with at least some matching data) on the left and right hand side to facilitate the join: `horizons(spc) <- newdata`

Arguments

object	A SoilProfileCollection
value	An object inheriting <code>data.frame</code>

Examples

```
# load test data
data(sp2)

# promote to SPC
depths(sp2) <- id ~ top + bottom

# assign true to surface horizon
newdata <- data.frame(top = 0,
                      newvalue = TRUE)

# do left join
horizons(sp2) <- newdata

# inspect site table: newvalue TRUE only for horizons
# with top depth equal to zero
horizons(sp2)
```

huePosition	<i>Munsell Hue Position for Soil Color Description</i>
-------------	--

Description

Munsell hues are typically arranged from 5R to 5PB in Munsell soil color books. This function matches a vector of Munsell hues to the position in this arrangement of 29 hues.

Usage

```
huePosition(x, returnHues = FALSE)
```

Arguments

x	character vector of hues, e.g. '10YR'
returnHues	logical, should the unique set of Munsell hues used for ordering be returned? See details.

Details

This function is fully vectorized.

Value

A vector of integer hue positions is typically returned, of the same length and order as x. If returnHues is TRUE, then the hue names and ordering is returned and x is ignored.

Author(s)

D.E. Beaudette

References

https://www.nrcs.usda.gov/wps/portal/nrcs/detail/soils/ref/?cid=nrcs142p2_053569

See Also

[colorContrast](#)

Examples

```
# get hue ordering for setting levels of a factor
huePosition(x=NULL, returnHues=TRUE)

# get position of the '10YR' hue (7)
huePosition(x='10YR')
```

hurst.redness	<i>Hurst (1977) Redness Index</i>
---------------	-----------------------------------

Description

Calculate Redness Index after Hurst (1977) "Visual estimation of iron in saprolite" DOI: 10.1130/0016-7606(1977)88<174:VEOIS>2.0.CO;2. Accepts vectorized inputs for hue, value and chroma, produces vector output.

Usage

```
hurst.redness(hue, value, chroma)
```

Arguments

hue	A character vector containing Munsell hues (e.g. "7.5YR")
value	A numeric vector containing Munsell values
chroma	A numeric vector containing Munsell chromas

Value

A numeric vector of horizon redness index (lower values = redder).

Author(s)

Andrew G. Brown.

References

Hurst, V.J. (1977) Visual estimation of iron in saprolite. GSA Bulletin. 88(2): 174–176. doi: [https://doi.org/10.1130/0016-7606\(1977\)88<174:VEOIS>2.0.CO;2](https://doi.org/10.1130/0016-7606(1977)88<174:VEOIS>2.0.CO;2)

HzDepthLogicSubset	<i>Subset SoilProfileCollection Objects or Horizons via checkHzDepthLogic</i>
--------------------	---

Description

This function removes profiles or horizons from a SoilProfileCollection that are flagged as having invalid horizon depth logic by [checkHzDepthLogic](#). Invalid profiles may be created when setting byhz = TRUE; use caution as some functions may not work properly in the presence of gaps. Consider using [fillHzGaps](#) to fill these gaps.

Usage

```
HzDepthLogicSubset(x, byhz = FALSE)
```

Arguments

x a SoilProfileCollection object
 byhz logical, evaluate horizon depth logic at the horizon level (profile level if FALSE)

Value

a SoilProfileCollection object

hzDepthTests *Tests of horizon depth logic*

Description

Function used internally by checkHzDepthLogic(), glom() and various other functions that operate on horizon data from single soil profiles and require a priori depth logic checks. Checks for bottom depths less than top depth / bad top depth order ("depthLogic"), bottom depths equal to top depth ("sameDepth"), overlaps/gaps ("overlapOrGap") and missing depths ("missingDepth"). Use names(res)[res] on result res of hzDepthTest() to determine type of logic error(s) found – see examples below.

Usage

```
hzDepthTests(top, bottom = NULL)
```

Arguments

top A numeric vector containing horizon top depths. Or a data.frame with two columns (first containing top depths, second containing bottom)
 bottom A numeric vector containing horizon bottom depths.

Value

A named logical vector containing TRUE for each type of horizon logic error found in the given data.

Author(s)

Andrew G. Brown & Dylan E. Beaudette

Examples

```
# no logic errors
res <- hzDepthTests(top = c(0,10,20,30), bottom = c(10,20,30,50))
names(res)[res]

# bottom < top
```

```

hzDepthTests(top = c(10,20,30,50), bottom = c(0,10,20,30))
names(res)[res]

# bottom == top
hzDepthTests(top = c(10,20,30,50), bottom = c(0,20,20,30))
names(res)[res]

# overlap
hzDepthTests(top = c(0,5,20,30), bottom = c(10,20,30,50))
names(res)[res]

# gap
hzDepthTests(top = c(0,15,20,30), bottom = c(10,20,30,50))
names(res)[res]

# missing
hzDepthTests(c(0,15,NA,30),c(10,NA,30,50))
names(res)[res]

```

hzDesgn

Get horizon designation column name

Description

Get horizon designation names

Arguments

object a SoilProfileCollection

hzdesgnname

Get horizon designation column name

Description

Get column name containing horizon designation name

Arguments

object a SoilProfileCollection

hzdesgnname<- *Set horizon designation column name*

Description

Set horizon designation column name

Arguments

object a SoilProfileCollection
value character, name of column containing horizon designations

Examples

```
data(sp1)

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

# set horizon designation column
hzdesgnname(sp1) <- "name"

# get horizon designation column
hzdesgnname(sp1)
```

hzDistinctnessCodeToOffset
 Convert Horizon Boundary Distinctness to Vertical Offset

Description

This function will convert USDA-NCSS horizon boundary distinctness codes into vertical (+/-) offsets in cm, based on the [Field Book for Describing and Sampling Soils, version 3.0](#).

Usage

```
hzDistinctnessCodeToOffset(  
  x,  
  codes = c("V", "A", "C", "G", "D"),  
  offset = c(0.5, 2, 5, 15, 20)/2  
)
```

Arguments

x	vector of boundary distinctness codes to be converted
codes	code values, adjust as needed
offset	vertical offset factors (cm), approximating 1/2 of the transitional zone thickness, see details

Details

The default offsets are based on the high-end of ranges presented in "transitional zone thickness criteria" from the Field Book version 3.0 (page 2-6). Offsets are returned as 1/2 of the transitional zone thickness so that horizon boundaries can be adjusted up/down from horizon depths. See [plotSPC](#), specifically the `hz.distinctness.offset` argument for visualization ideas. Missing data in `x` (NA) or codes that are not defined in `codes` are returned as 0 offsets.

Additional examples are available in the [Visualization of Horizon Boundaries tutorial](#).

Value

vector of offsets with same length as `x`

Author(s)

D.E. Beaudette

References

[Field Book for Describing and Sampling Soils, version 3.0](#)

See Also

[plotSPC](#)

Examples

```
# example data
data(sp1)

# compute 1/2 transitional zone thickness from distinctness codes
sp1$hzdo <- hzDistinctnessCodeToOffset(sp1$bound_distinct)

# 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

# adjust margins
op <- par(mar=c(0,0,0,1.5))

# sketches, adjust width, adjust text size, include coded hz distinctness offsets
plotSPC(sp1, width=0.3, cex.names=0.75, hz.distinctness.offset = 'hzdo')
```

```
# clean-up
par(op)
```

```
hzID          Get horizon IDs
```

Description

Get vector containing horizon IDs

Arguments

object a SoilProfileCollection

```
hzID<-       Set horizon IDs
```

Description

Set vector containing horizon IDs

Arguments

object a SoilProfileCollection
value a unique vector of equal length to number of horizons nrow(object)

```
hzidname<-   Set horizon ID column name
```

Description

Set unique horizon ID column name
Get column name containing unique horizon ID

Arguments

value character, column name containing unique horizon ID values
object a SoilProfileCollection

Examples

```

data(sp1)

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

# create new horizon ID
sp1$hziDrev <- rev(sp1$hziD)

# set horizon designation column
hziDname(sp1) <- "hziDrev"

# get horizon designation column
hziDname(sp1)

```

hztexclname	<i>Get horizon texture class column name</i>
-------------	--

Description

Get column name containing horizon designation name

Arguments

object	a SoilProfileCollection
--------	-------------------------

hztexclname<-	<i>Set horizon texture class column name</i>
---------------	--

Description

Set horizon texture class column name for a SoilProfileCollection

Arguments

object	a SoilProfileCollection
value	character, name of column containing horizon texture classes

Examples

```
data(sp1)

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

# set horizon texture class column
hztexclname(sp1) <- "texture"

# get horizon texture class column
hztexclname(sp1)
```

hzTopographyCodeToLineType

Convert Horizon Boundary Topography to Line Type

Description

This function will convert USDA-NCSS horizon boundary topography codes into line types, based on the [Field Book for Describing and Sampling Soils, version 3.0](#).

Usage

```
hzTopographyCodeToLineType(  
  x,  
  codes = c("S", "W", "I", "B"),  
  lty = c(1, 2, 3, 4)  
)
```

Arguments

x	vector of boundary topography codes to be converted
codes	code values, adjust as needed
lty	line types

Details

Visualization of horizon boundary topography can be difficult, line type offers an additional visual cue. See `hzTopographyCodeToOffset` for an offset-based approach. Additional examples are available in the [Visualization of Horizon Boundaries tutorial](#).

Value

vector of line types with same length as x

Author(s)

D.E. Beaudette

References[Field Book for Describing and Sampling Soils, version 3.0](#)**See Also**[plotSPC](#), [hzTopographyCodeToOffset](#)

`hzTopographyCodeToOffset`*Convert Horizon Boundary Topography to Vertical Offset*

Description

This function will convert USDA-NCSS horizon boundary topography codes into a vertical offset, suitable for use in `plotSPC`. Default values are reasonable starting points for encoding smooth, wavy, irregular, or broken style horizon boundary topography as defined in [Field Book for Describing and Sampling Soils, version 3.0](#).

Usage

```
hzTopographyCodeToOffset(  
  x,  
  codes = c("S", "W", "I", "B"),  
  offset = c(0, 4, 8, 12)  
)
```

Arguments

<code>x</code>	vector of boundary topography codes to be converted
<code>codes</code>	code values, adjust as needed
<code>offset</code>	vertical offset (depth units) used to create "chevron" effect

Details

Additional examples are available in the [Visualization of Horizon Boundaries tutorial](#).

Value

vector of vertical offsets with same length as `x`

Author(s)

D.E. Beaudette

References

[Field Book for Describing and Sampling Soils, version 3.0](#)

See Also

[plotSPC](#)

hzTransitionProbabilities

Horizon Transition Probabilities

Description

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

Usage

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

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.

Details

See the following tutorials for some ideas:

horizon designation TP <http://ncss-tech.github.io/AQP/aqp/hz-transition-probabilities.html>

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

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, t_0 . 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)
```

idname	<i>Get profile ID column name</i>
--------	-----------------------------------

Description

Get column name containing unique profile IDs

Arguments

object	a SoilProfileCollection
--------	-------------------------

invertLabelColor	<i>Make High Contrast Label Colors</i>
------------------	--

Description

Generate a vector of white or black label colors conditioned on a vector of colors to maximize label contrast.

Usage

```
invertLabelColor(colors, threshold = 0.65)
```

Arguments

colors	vector of colors
threshold	black white threshold

Value

vector of label colors

Author(s)

D.E. Beaudette

 jacobs2000

Soil Morphologic Data from Jacobs et al. 2002.

Description

Select soil morphologic data from "Redoximorphic Features as Indicators of Seasonal Saturation, Lowndes County, Georgia". This is a useful sample dataset for testing the analysis and visualization of redoximorphic features.

Usage

```
data(jacobs2000)
```

Format

A SoilProfileCollection object.

References

Jacobs, P. M., L. T. West, and J. N. Shaw. 2002. Redoximorphic Features as Indicators of Seasonal Saturation, Lowndes County, Georgia. *Soil Sci. Soc. Am. J.* 66:315-323. doi:10.2136/sssaj2002.3150

Examples

```
# load
data(jacobs2000)

# basic plot
par(mar=c(0,1,3,3))
plot(jacobs2000, name='name', color='matrix_color', width=0.3)
# add concentrations
addVolumeFraction(jacobs2000, 'concentration_pct',
col = jacobs2000$concentration_color, pch = 16, cex.max = 0.5)

# add depletions
plot(jacobs2000, name='name', color='matrix_color', width=0.3)
addVolumeFraction(jacobs2000, 'depletion_pct',
col = jacobs2000$depletion_color, pch = 16, cex.max = 0.5)

# time saturated
plotSPC(jacobs2000, color='time_saturated', cex.names=0.8, col.label = 'Time Saturated')

# color contrast: matrix vs. concentrations
cc <- colorContrast(jacobs2000$matrix_color_munsell, jacobs2000$concentration_munsell)
cc <- na.omit(cc)

cc <- cc[order(cc$dE00), ]
cc <- unique(cc)
```

```

par(bg='black', fg='white')
colorContrastPlot(cc$m1[1:10], cc$m2[1:10], labels = c('matrix', 'concentration'))
colorContrastPlot(cc$m1[11:21], cc$m2[11:21], labels = c('matrix', 'concentration'))

# color contrast: depletion vs. concentrations
cc <- colorContrast(jacobs2000$depletion_munsell, jacobs2000$concentration_munsell)
cc <- na.omit(cc)

cc <- cc[order(cc$dE00), ]
cc <- unique(cc)

par(bg='black', fg='white')
colorContrastPlot(cc$m1, cc$m2, labels = c('depletion', 'concentration'))

```

L1_profiles

Create Representative Soil Profiles via L1 Estimator

Description

The L1 estimator, or **geometric median**, is a multivariate generalization of the (univariate) **median** concept. This function performs a multivariate aggregation (via L1 estimator) according to a suite of ratio-scale soil properties. The L1 estimator is applied to soil profile data that have been sliced to a 1-depth-unit basis.

See the [L1 Profiles Tutorial](#) for additional examples.

Usage

```

L1_profiles(
  x,
  fm,
  basis = 1,
  method = c("regex", "simple", "constant"),
  maxDepthRule = c("max", "min"),
  maxDepthConstant = NULL,
  strict = FALSE
)

```

Arguments

x	SoilProfileCollection object
fm	formula, for example: group ~ p1 + p2 + p3, where "group" is a site-level grouping variable, and "p1", "p2", and "p3" are horizon level variables
basis	positive integer, aggregation basis (e.g. 1 for 1-depth-unit intervals). Values other than 1 are not currently supported.

method	soil depth evaluation method: "regex" for regular expression, "simple", or "constant". See details.
maxDepthRule	maximum depth rule: "max" or "min" See details.
maxDepthConstant	positive integer, maximum depth when maxDepthRule = 'constant'
strict	passed to slice

Details

See [this related tutorial](#) for additional examples. The method, maxDepthRule, and maxDepthConstant arguments set the maximum depth (over the entire collection) of analysis used to build "L1 profiles". The following rules are available:

- method = 'regex' uses pattern matching on horizon designations (note that hzdesgnname metadata must be set with hzdesgnname(x) <- 'columnname')
- method = 'simple' uses min or max as applied to x, no accounting for non-soil horizons (e.g. Cr or R)
- method = 'constant' uses a fixed depth value supplied by maxDepthConstant

The maxDepthRule argument sets depth calculation constraint, applied to soil depths computed according to method (min or max).

Value

a SoilProfileCollection object

Note

This function requires the Gmedian package.

References

Cardot, H., Cenac, P. and Zitt, P-A. (2013). Efficient and fast estimation of the geometric median in Hilbert spaces with an averaged stochastic gradient algorithm. *Bernoulli*, 19, 18-43.

length	<i>Get the number of profiles in a SoilProfileCollection</i>
--------	--

Description

Get the number of profiles in a SoilProfileCollection

Usage

```
## S4 method for signature 'SoilProfileCollection'
length(x)
```


Arguments

x a SoilProfileCollection

lunique	<i>Eliminate duplicate instances of profile IDs in a list of SoilProfileCollections</i>
---------	---

Description

@description Experimental function to "clean" list input where duplicates exist (that would otherwise prevent pbindlist). Useful for queries that may have overlapping instances of the same data, for instance a list of SoilProfileCollections where each list element contains profiles gathered from a set of (potentially overlapping) extents.

Usage

```
lunique(l)
```

Arguments

l A list of SoilProfileCollections.

Value

A list of SoilProfileCollections, with duplicate profile IDs removed.

Author(s)

Andrew G. Brown

Examples

```
data(sp5)

# EXAMPLE #1 -- resolving overlap

# 6 profiles in four sets, and 5,6,7 are missing
input <- lapply(list(c(1,3,4), c(2,2,3), NA, c(8,9,1)), function(idx) {
  if(!all(is.na(idx)))
    sp5[idx,]
})

output <- lunique(input)

# 6 profiles are in final SPC; 5,6,7 are missing
match(profile_id(pbindlist(output)), profile_id(sp5))

# EXAMPLE #2 -- exact duplicates
```

```
# deliberately duplicate an SPC
sp5_2 <- sp5
res <- lunique(list(sp5, sp5_2))

# the number of profiles in first element is equal to number in sp5
length(res[[1]]) == length(sp5)

# second list element contains NA b/c all uniques are in #1
res[[2]]
```

max	<i>Get the maximum bottom depth in a SoilProfileCollection</i>
-----	--

Description

Get the deepest depth of description out of all profiles in a SoilProfileCollection. Data missing one or more of: bottom depth, profile ID, or any optional attribute are omitted using `complete.cases`.

Usage

```
## S4 method for signature 'SoilProfileCollection'
max(x, v = NULL)
```

Arguments

x	a SoilProfileCollection
v	optional: horizon-level column name to refine calculation

metadata	<i>Retrieve metadata from SoilProfileCollection</i>
----------	---

Description

Get metadata from SoilProfileCollection. Result is a list. Two entries (`aqp_df_class`, `depth_units`) should not be edited in the metadata list directly. There are methods that facilitate changing them – and propagating their changes throughout the collection. Otherwise, metadata list is a free-form slot used to store arbitrary information about the data, how it was collected, citations, etc.

Usage

```
## S4 method for signature 'SoilProfileCollection'
metadata(object)

metadata(object) <- value
```

Arguments

object	A SoilProfileCollection
value	A named list (see examples)

Examples

```
data(sp5)

# replace default metadata with itself
metadata(sp5) <- metadata(sp5)

# set new metadata attribute value
metadata(sp5)$newvalue <- 'foo'

# get metadata attribute
metadata(sp5)$newvalue
```

min

Get the minimum bottom depth in a SoilProfileCollection

Description

Get the shallowest depth of description out of all profiles in a SoilProfileCollection. Data missing one or more of: bottom depth, profile ID, or any optional attribute are omitted using `complete.cases`.

Usage

```
## S4 method for signature 'SoilProfileCollection'
min(x, v = NULL)
```

Arguments

x	a SoilProfileCollection
v	optional: a vector of horizon attribute names to refine calculation

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**

```

# 10 random profiles
set.seed(10101)
s <- lapply(as.character(1:10), random_profile)
s <- do.call('rbind', s)

# 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'
)

```

mixMunsell

Mix Munsell Colors via Spectral Library

Description

Simulate mixing of colors in Munsell notation, similar to the way in which mixtures of pigments operate.

Usage

```

mixMunsell(
  x,
  w = rep(1, times = length(x))/length(x),
  mixingMethod = c("spectra", "estimate", "adaptive"),
  n = 1,
  keepMixedSpec = FALSE,
  distThreshold = 0.025
)

```

Arguments

x	vector of colors in Munsell notation
w	vector of proportions, can sum to any number
mixingMethod	approach used to simulate a mixture: <ul style="list-style-type: none"> • spectra: simulate a subtractive mixture of pigments, limited to available reference spectra • estimate: closest Munsell chip to a weighted mean of CIELAB coordinates • adaptive: use reference spectra when possible, falling-back to weighted mean of CIELAB coordinates
n	number of closest matching color chips (mixingMethod = spectra only)
keepMixedSpec	keep weighted geometric mean spectra, final result is a list (mixingMethod = spectra only)
distThreshold	spectral distance used to compute scaledDistance, default value is based on an analysis of spectral distances associated with adjacent Munsell color chips.

Details

An accurate simulation of pigment mixtures ("subtractive" color mixtures) is incredibly complex due to factors that aren't easily measured or controlled: pigment solubility, pigment particle size distribution, water content, substrate composition, and physical obstruction to name a few. That said, it is possible to simulate reasonable, subtractive color mixtures given a reference spectra library (350-800nm) and some assumptions about pigment qualities and lighting. For the purposes of estimating a mixture of soil colors (these are pigments after all) we can relax these assumptions and assume a standard light source. The only missing piece is the spectral library for all Munsell chips in our color books.

Thankfully, [Scott Burns has outlined the entire process](#), and Paul Centore has provided a Munsell color chip [reflectance spectra library](#). The estimation of a subtractive mixture of soil colors can proceed as follows:

1. look up the associated spectra for each color
2. computed the weighted (area proportion) geometric mean of the spectra
3. search for the closest matching spectra in the reference library
4. suggest that Munsell chip as the best candidate for the simulated mixture

Key assumptions include:

- similar particle size distribution
- similar mineralogy (i.e. pigmentation qualities)
- similar water content.

For the purposes of estimating (for example) a "mixed soil color within the top 18cm of soil" these assumptions are usually valid. Again, these are estimates that are ultimately "snapped" to the nearest chip and do not need to approach the accuracy of paint-matching systems.

A message is printed when scaledDistance is larger than 1.

Value

A data.frame with the closest matching Munsell color(s):

- munsell: Munsell notation of the n-closest spectra
- distance: spectral (Gower) distance to the n-closest spectra
- scaledDistance: spectral distance scaled by distThreshold
- mixingMethod: method used for each mixture

When keepMixedSpec = TRUE then a list:

- mixed: a data.frame containing the same elements as above
- spec: spectra for the 1st closest match

Author(s)

D.E. Beaudette

References

- inspiration / calculations based on the work of Scott Burns: <https://arxiv.org/ftp/arxiv/papers/1710/1710.06364.pdf>
- related discussion on Stack Overflow: <https://stackoverflow.com/questions/10254022/implementing-kubelka-munk-like-krita-to-mix-colours-color-like-paint/29967630#29967630>
- spectral library source: <https://www.munsellcolourscienceforpainters.com/MunsellResources/SpectralReflectancesOf2007MunsellBookOfColorGlossy.txt>

See Also

[munsell.spectra](#)

mollic.thickness.requirement

Calculate the minimum thickness requirement for Mollic epipedon

Description

Utilize horizon depths, designations and textures in a profile to estimate the thickness requirement for the Mollic or Umbric epipedon, per criterion 6 in the U.S. Keys to Soil Taxonomy (12th Edition).

Usage

```
mollic.thickness.requirement(
  p,
  texcl.attr = guessHzTexClName(p),
  clay.attr = guessHzAttrName(p, "clay", c("total", "_r")),
  truncate = TRUE
)
```

Arguments

<code>p</code>	A single-profile SoilProfileCollection.
<code>texcl.attr</code>	Column in horizon table containing texture classes. Default: <code>guessHzTexClName(p)</code>
<code>clay.attr</code>	Column in horizon table containing clay contents. Default: <code>guessHzAttrName(p, 'clay', c('total', '_'))</code>
<code>truncate</code>	Should sliding scale (Criterion 6C) results be truncated to 18 to 25cm interval? (Experimental; Default: TRUE)

Value

A unit length numeric vector containing Mollic or Umbric epipedon minimum thickness requirement.

Author(s)

Andrew G. Brown

Examples

```
# construct a fake profile
spc <- data.frame(id=1, taxsubgrp = "Lithic Haploxeralfs",
  hzname   = c("A", "AB", "Bt", "BCt", "R"),
  hzdept   = c(0, 20, 32, 42, 49),
  hzdepb   = c(20, 32, 42, 49, 200),
  prop     = c(18, 22, 28, 24, NA),
  texcl    = c("1", "1", "cl", "1", "br"),
  d_value  = c(5, 5, 5, 6, NA),
  m_value  = c(2.5, 3, 3, 4, NA),
  m_chroma = c(2, 3, 4, 4, NA))

# promote to SoilProfileCollection
depths(spc) <- id ~ hzdept + hzdepb
hzdesgname(spc) <- 'hzname'
hztexclname(spc) <- 'texcl'

# print results in table
data.frame(id = spc[[idname(spc)]],
  thickness_req = mollic.thickness.requirement(spc, clay.attr='prop'),
  thickness_req_nobound = mollic.thickness.requirement(spc,
    clay.attr='prop', truncate=FALSE))
```

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)
```

`munsell.spectra`*Spectral Library of Munsell Colors*

Description

The original database "SpectralReflectancesOf2007MunsellBookOfColorGlossy.txt" was provided by Paul Centore and downloaded July, 2020. Reflectance values for odd chroma have been interpolated from adjacent chips. See `aqp/misc/utills/Munsell/` for the entire set of processing steps.

Munsell value typically ranges from 2-9, and chroma from 1-12. Ranges vary by hue. Run `aqp:::summarizeMunsellSpectraRanges()` for a detailed listing by hue.

The original database contains the following description:

This file contains spectral reflectance measurements of X-Rite's 2007 Munsell Book of Color (Glossy Finish). The measurements were made in 2012 with a ColorMunki spectrophotometer. The first column is the Munsell name. The remaining columns give reflectance values for 380 nm to 730 nm, in steps of 10 nm. The reflectance is a value between 0 (indicating that no light at that wavelength is reflected) and 1 (indicating that all the light at that wavelength is reflected). Occasionally an entry is slightly greater than 1. The likely cause is random variability, and those entries can be adjusted to 1 with negligible loss. In all, 1485 colour samples were measured. Researchers are invited to analyze the data in this file.

Usage

```
data(munsell.spectra)
```

Format

A data frame with 89496 rows and 10 variables:

munsell munsell color

hue hue component

value value component

chroma chroma component

wavelength wavelength (nm)

reflectance reflectance

References

Centore, Paul. Colour Tools for Painters. <https://www.munsellcolourscienceforpainters.com/>.

munsell2rgb	<i>Convert Munsell Color Notation to other Color Space Coordinates (sRGB and CIELAB)</i>
-------------	--

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,
  returnLAB = FALSE
)
```

Arguments

the_hue	a vector of one or more more hues, upper-case (e.g. '10YR')
the_value	a vector of one or more values (e.g. '4')
the_chroma	a vector of one or more chromas (e.g. '6'), may be NA for neutral hues
alpha	numeric, transparency setting used when return_triplets = FALSE and returnLAB = FALSE
maxColorValue	maximum sRGB color value, typically 1 (see rgb)
return_triplets	logical, return sRGB coordinates (range 0-1) instead of standard hex notation of sRGB (e.g. '#8080B')
returnLAB	logical, return CIELAB coordinates (D65 illuminant)

Details

This function is vectorized without recycling: i.e. the length of each argument must be 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\

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 notation (e.g. '7.9YR 2.7/2.0') can be matched (nearest-neighbor, no interpolation) to the closest color within the munsell sRGB/CIELAB look-up table via `getClosestMunsellChip()`. A more accurate estimate of sRGB values from non-standard notation can be achieved with the [munsellinterpol](#) package.

See examples below.

Value

A vector of R colors is returned that is the same length as the input data. When `return_triplets = TRUE` and/or `returnLAB = TRUE`, then a `data.frame` (of sample length as input) 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> <https://www.munsellcolourscienceforpainters.com/MunsellAndKubelkaMunkToolbox/MunsellAndKubelkaMunkToolbox.html>

Examples

```
# 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
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)
# convert Munsell -> sRGB
d$color <- with(d, munsell2rgb(hue, value, chroma))

# extract CIELAB coordinates
with(d, munsell2rgb(hue, value, chroma, returnLAB=TRUE))

# 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])
}
)

# 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')

```

```
munsell2spc,SoilProfileCollection-method
```

*Merge Munsell Hue, Value, Chroma converted to sRGB & CIELAB
into a SoilProfileCollection*

Description

Convert Munsell hue, value and chroma into sRGB (rgb_R, rgb_G, rgb_B) and CIELAB (lab_L, lab_A, lab_B) color coordinates using munsell2rgb. The converted values are stored in the horizons() slot unless as.spc is FALSE, in which case the results are combined with profile and horizon ID columns and returned as the data.frame subclass used by the SPC.

Usage

```
## S4 method for signature 'SoilProfileCollection'
munsell2spc(
```

```

object,
hue = "hue",
value = "value",
chroma = "chroma",
.data = NULL,
as.spc = TRUE
)

```

Arguments

object	A SoilProfileCollection
hue	Column name containing numeric hue values. Default: "hue"
value	Column name containing numeric value values. Default: "value"
chroma	Column name containing numeric chroma values. Default: "chroma"
.data	Optional: a character vector of equal length to number of horizons (containing Munsell notation), or a column name in the horizon data OR a data.frame containing three columns (names specified in hue, value, chroma)
as.spc	Return a data.frame-like object with ID columns?

Value

A SoilProfileCollection or data.frame-like object

See Also

[parseMunsell](#) [rgb2munsell](#) [munsell2rgb](#)

Examples

```

data(sp3)
depths(sp3) <- id ~ top + bottom

# inspect input data
horizons(sp3)[,c("hue", "value", "chroma")]

# do color conversions to sRGB and LAB, join into horizon data
sp3 <- munsell2spc(sp3)

# plot rgb "R" coordinate by horizon
plot(sp3, color = "rgb_R")

# plot lab "A" coordinate by horizon
plot(sp3, color = "lab_A")

# note that `lab_A` values do not exactly match the original `A` values
# this is because `lab_A` was computed from the (field determined) Munsell color notation,
# while `A` was directly measured in the lab by colorimeter
plot(sp3$A, sp3$lab_A, xlab = 'Measured', ylab = 'Converted from Field Observed Munsell')

```

mutate_profile	<i>Transform a SPC (by profile) with a set of expressions</i>
----------------	---

Description

mutate_profile() is a function used for transforming SoilProfileCollections. Each expression is applied to site or horizon level attributes of individual profiles. This distinguishes this function from mutate, which is applied to pooled values (across individuals) in a collection/group.

Arguments

object	A SoilProfileCollection
...	A set of comma-delimited R expressions that resolve to a transformation to be applied to a single profile e.g mutate_profile(hzdept = max(hzdept) - hzdept)

Value

A SoilProfileCollection.

Author(s)

Andrew G. Brown.

names	<i>Get names of columns in site and horizons table</i>
-------	--

Description

Get names of columns in site and horizons table of a SoilProfileCollection.

Usage

```
## S4 method for signature 'SoilProfileCollection'  
names(x)
```

Arguments

x	a SoilProfileCollection
---	-------------------------

nrow	<i>Get the number of horizons in a SoilProfileCollection</i>
------	--

Description

Get the number of horizons in a SoilProfileCollection

Arguments

object	a SoilProfileCollection
--------	-------------------------

overlapMetrics	<i>Find and Quantify Overlap within a 1D Sequence</i>
----------------	---

Description

Desc.

Usage

```
overlapMetrics(x, thresh)
```

Arguments

x	vector of relative horizontal positions, one for each profile
thresh	threshold defining "overlap", typically < 1

@return a list:

- idx: unique index to overlapping elements in x
- ov: normalized overlap (see details)

Examples

```
x <- c(1, 2, 3, 3.4, 3.5, 5, 6, 10)
overlapMetrics(x, thresh = 0.5)
```

panel.depth_function *Lattice Panel Function for Soil Profiles*

Description

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

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`.

This function is not able to apply transformations (typically $\log = 10$) applied in the `scales` argument to `xypplot` to upper/lower bounds. These will have to be manually applied. See examples.

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

<code>x</code>	x values (generated by calling lattice function)
<code>y</code>	y values (generated by calling lattice function)
<code>id</code>	vector of id labels, same length as x and y—only required when plotting segments (see Details section)
<code>upper</code>	vector of upper confidence envelope values
<code>lower</code>	vector of lower confidence envelope values
<code>subscripts</code>	paneling indices (generated by calling lattice function)
<code>groups</code>	grouping data (generated by calling lattice function)
<code>sync.colors</code>	optionally sync the fill color within the region bounded by (lower–upper) with the line colors
<code>cf</code>	optionally annotate contributing fraction data at regular depth intervals see slab

cf.col optional 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

Author(s)

D.E. Beaudette

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'
    )))
  )

# more complex examples, using step functions with grouped data
# better looking figures with less customization via tactile package
if(requireNamespace('tactile')) {

  library(data.table)
  library(lattice)
  library(tactile)

  # example data
  data(sp6)

  # a single profile
  x <- sp6[1:5, ]

  # wide -> long format
  x.long <- melt(
    data.table(x),
    id.vars = c('id', 'top', 'bottom'),
    measure.vars = c('sand', 'silt', 'clay')
  )

  # (optional) convert back to data.frame
  x.long <- as.data.frame(x.long)

  # three variables sharing a common axis
  # factor levels set by melt()
  xyplot(
    cbind(top, bottom) ~ value | id,
    groups = variable,
    data = x.long,
    id = x.long$id,
    ylim = c(200, -5), xlim = c(10, 60),
    scales = list(alternating = 1, y = list(tick.number = 10)),
    par.settings = tactile.theme(superpose.line = list(lwd = 2)),
    xlab = 'Sand, Silt, Clay (%)',
    ylab = 'Depth (cm)',
    panel = panel.depth_function,
    auto.key = list(columns = 3, lines = TRUE, points = FALSE),
    asp = 1.5
  )

  # all profiles
  x <- sp6

  # wide -> long format
  x.long <- melt(
    data.table(x),

```

```

    id.vars = c('id', 'top', 'bottom'),
    measure.vars = c('sand', 'silt', 'clay')
  )

  # (optional) convert back to data.frame
  x.long <- as.data.frame(x.long)

  # three variables sharing a common axis
  # factor levels set by melt()
  xyplot(
    cbind(top, bottom) ~ value | id,
    groups = variable,
    data = x.long,
    id = x.long$id,
    ylim = c(200, -5), xlim = c(0, 70),
    scales = list(alternating = 1, y = list(tick.number = 10)),
    par.settings = tactile.theme(superpose.line = list(lwd = 2)),
    xlab = 'Sand, Silt, Clay (%)',
    ylab = 'Depth (cm)',
    panel = panel.depth_function,
    auto.key = list(columns = 3, lines = TRUE, points = FALSE),
    as.table = TRUE
  )

  xyplot(
    cbind(top, bottom) ~ value,
    groups = variable,
    data = x.long,
    id = x.long$id,
    ylim = c(200, -5), xlim = c(0, 70),
    scales = list(alternating = 1, y = list(tick.number = 10)),
    par.settings = tactile.theme(superpose.line = list(lwd = 2)),
    xlab = 'Sand, Silt, Clay (%)',
    ylab = 'Depth (cm)',
    panel = panel.depth_function,
    auto.key = list(columns = 3, lines = TRUE, points = FALSE),
    as.table = TRUE
  )

  xyplot(
    cbind(top, bottom) ~ value | variable,
    groups = variable,
    data = x.long,
    id = x.long$id,
    ylim = c(200, -5), xlim = c(0, 70),
    scales = list(alternating = 1, y = list(tick.number = 10)),
    par.settings = tactile.theme(superpose.line = list(lwd = 2)),
    xlab = 'Sand, Silt, Clay (%)',
    ylab = 'Depth (cm)',
    panel = panel.depth_function,
    auto.key = list(columns = 3, lines = TRUE, points = FALSE),
    as.table = TRUE
  )

```

```

xyplot(
  cbind(top, bottom) ~ value | variable,
  data = x.long,
  id = x.long$id,
  ylim = c(200, -5), xlim = c(0, 70),
  scales = list(alternating = 1, y = list(tick.number = 10)),
  par.settings = tactile.theme(superpose.line = list(lwd = 2)),
  xlab = 'Sand, Silt, Clay (%)',
  ylab = 'Depth (cm)',
  panel = panel.depth_function,
  auto.key = list(columns = 3, lines = TRUE, points = FALSE),
  as.table = TRUE
)
}

```

 parseMunsell

Parse Munsell Color Notation

Description

Split Munsell color notation into "hue", "value", and "chroma", with optional conversion to sRGB hex notation, sRGB coordinates, and CIELAB coordinates. Conversion is performed by `munsell2rgb`.

Usage

```
parseMunsell(munsellColor, convertColors = TRUE, ...)
```

Arguments

<code>munsellColor</code>	character vector of Munsell colors (e.g. <code>c('10YR 3/4', '5YR 4/6')</code>)
<code>convertColors</code>	logical, convert colors to sRGB hex notation, sRGB coordinates, CIELAB coordinates
<code>...</code>	additional arguments to <code>munsell2rgb</code>

Value

a `data.frame` object

Examples

```
# just sRGB
parseMunsell("10YR 3/5", return_triplets = TRUE)

# sRGB + CIELAB (D65 illuminant)
parseMunsell("10YR 3/5", return_triplets = TRUE, returnLAB = TRUE)

# CIELAB only
parseMunsell("10YR 3/5", return_triplets = FALSE, returnLAB = TRUE)
```

pbindlist

Combine a list of SoilProfileCollection objects

Description

See `combine(...)` for a connotative short-hand method that does not require that `SoilProfileCollection` be in a list. Profiles will be sorted based on character sorting of profile ID.

Usage

```
pbindlist(l, new.idname = NULL, verbose = TRUE)
```

Arguments

<code>l</code>	a list of <code>SoilProfileCollection</code> objects
<code>new.idname</code>	Optional: a character referring to a new column name to put unique profile IDs in; default: <code>NULL</code> to attempt with existing <code>idname</code> in first element
<code>verbose</code>	Produce warnings and messages regarding results? default: <code>TRUE</code>

Details

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. If one or more subsequent list elements has non-unique values in a site level attribute of that name, the ID name from the second list element is attempted, and so on. Non-conforming spatial data are dropped from the final result (returns default empty `SpatialPoints`).

Value

a `SoilProfileCollection` object

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 <- pbindlist(list(x, y))

# check
plot(z)
```

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

Author(s)

Dylan 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, 2013, Pages 258-268, ISSN 0098-3004, doi: [10.1016/j.cageo.2012.10.020](https://doi.org/10.1016/j.cageo.2012.10.020).
- 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:
library(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', asp=1.5,
       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(...)
    }
)

# more soil properties
data(sp2)
depths(sp2) <- id ~ top + bottom

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
set.seed(1010101)
s <- pbindlist(lapply(letters[1:10], random_profile, SPC=TRUE))

# 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)) {

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

  d <- profile_compare(sp2, vars=c('prop','field_ph','hue','value'),
                      max_d=100, k=0)

  h <- diana(d)
  p <- as.phylo(as.hclust(h))

```

```

plot(p, show.tip.label=FALSE)
tiplabels(sp2$surface, col=cutree(h, 3), bg=NA, cex=0.75)

## 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 successively 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)

```

perturb

Perturb soil horizon depths using boundary distinctness

Description

"Perturbs" the **boundary between horizons** or the **thickness of horizons** using a standard deviation specified as a horizon-level attribute. This is selected using either `boundary.attr` or `thickness.attr` to specify the column name.

The boundary standard deviation corresponds roughly to the concept of "horizon boundary distinctness." In contrast, the *horizon thickness* standard deviation corresponds roughly to the "variation in horizon thickness" so it may be determined from several similar profiles that have a particular layer "in common."

Usage

```

perturb(
  p,
  n = 100,
  id = NULL,
  thickness.attr = NULL,
  boundary.attr = NULL,
  min.thickness = 1,
  max.depth = NULL,
  new.idname = "pID"
)

```

Arguments

<code>p</code>	A single-profile SoilProfileCollection
<code>n</code>	Number of new profiles to generate (default: 100)
<code>id</code>	a vector of profile IDs with length equal to (n). Overrides use of <code>seq_len(n)</code> as default profile ID values.
<code>thickness.attr</code>	Horizon variance attribute containing numeric "standard deviations" reflecting horizon thickness
<code>boundary.attr</code>	Horizon variance attribute containing numeric "standard deviations" reflecting boundary transition distinctness
<code>min.thickness</code>	Minimum thickness of permuted horizons (default: 1)
<code>max.depth</code>	Depth below which horizon depths are not perturbed (default: NULL)
<code>new.idname</code>	New column name to contain unique profile ID (default: pID)

Details

Imagine a Normal curve with mean centered on the vertical (depth axis) at a representative value (RV) horizon bottom depth or thickness. By the Empirical Rule for Normal distribution, two "standard deviations" above or below that "central" mean value represent 95% of the "typical volume" of that horizon or boundary.

`perturb` can leverage semi-quantitative (ordered factor) levels of boundary distinctness/topography for the upper and lower boundary of individual horizons. A handy function for this is [hzDistinctnessCodeToOffset\(\)](#). The `boundary.attr` is arguably easier to parameterize from a single profile description or "Form 232" where *horizon boundary distinctness* classes (based on vertical distance of transition) are conventionally recorded for each layer.

Alternately, `perturb` can be parameterized using standard deviation in thickness of layers derived from a group. Say, the variance parameters are defined from a set of pedons correlated to a particular series or component, and the template "seed" profile is, for example, the Official Series Description or the Representative Component Pedon.

Value

a SoilProfileCollection with n realizations of p

Author(s)

D.E. Beaudette, A.G. Brown

See Also

[random_profile\(\)](#) [hzDistinctnessCodeToOffset\(\)](#)

Examples

```
### THICKNESS

# 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
horizons(s)$hz.sd <- 2 # constant standard deviation
sim.1 <- perturb(s, n = 25, thickness.attr = "hz.sd")

# simulate 25 new profiles using different SD for each horizon
horizons(s)$hz.sd <- c(1, 2, 5, 5, 5, 10, 3)
sim.2 <- perturb(s, n = 25, thickness.attr = "hz.sd")

# 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, 3)',
  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(data.table)
a.long <- melt(as.data.table(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$top, lty = 2, lwd = 2)
  }
)

### BOUNDARIES

# example with sp1 (using boundary distinctness)
data("sp1")
depths(sp1) <- id ~ top + bottom

# specify "standard deviation" for boundary thickness
# consider a normal curve centered at boundary RV depth
# lookup table: ~maximum thickness of boundary distinctness classes, divided by 3
bound.lut <- c('V'=0.5, 'A'=2, 'C'=5, 'G'=15, 'D'=45) / 3

## V      A      C      G      D
## 0.1666667 0.6666667 1.6666667 5.0000000 15.0000000

sp1$bound_sd <- bound.lut[sp1$bound_distinct]

# hold any NA boundary distinctness constant
sp1$bound_sd[is.na(sp1$bound_sd)] <- 0

quantile(sp1$bound_sd, na.rm = TRUE)

```

```
p <- sp1[3]

# assume boundary sd is 1/12 midpoint of horizon depth
# (i.e. general relationship: SD increases (less well known) with depth)
sp1 <- transform(sp1, midpt = (bottom - top) / 2 + top, bound_sd = midpt / 12)
quantile(sp1$bound_sd)

perturb(p, boundary.attr = "bound_sd")
```

plotColorMixture

Visualize Spectral Mixing of Munsell Colors

Description

Lattice visualization demonstrating subtractive mixtures of colors in Munsell notation and associated spectra.

Usage

```
plotColorMixture(
  x,
  w = rep(1, times = length(x))/length(x),
  n = 1,
  swatch.cex = 6,
  label.cex = 0.85,
  showMixedSpec = FALSE,
  overlapFix = TRUE
)
```

Arguments

x	vector of colors in Munsell notation, should not contain duplicates
w	vector of proportions, can sum to any number
n	number of closest mixture candidates (see mixMunsell), results can be hard to interpret
swatch.cex	scaling factor for color swatch
label.cex	scaling factor for swatch labels
showMixedSpec	show weighted geometric mean (mixed) spectra as dotted line
overlapFix	attempt to "fix" overlapping chip labels via fixOverlap

Value

lattice graphics object

Author(s)

D.E. Beaudette

plotColorQuantiles *Visualize Color Quantiles*

Description

This function creates a visualization of the output from colorQuantiles using lattice graphics.

Usage

```
plotColorQuantiles(res, pt.cex = 7, lab.cex = 0.66)
```

Arguments

res	list returned by colorQuantiles
pt.cex	scaling factor for color chips
lab.cex	chip label scaling factor

Details

Marginal percentiles and L1 median CIELAB values from colorQuantiles are combined into a single plot, arranged in panels according to L, A, and B coordinates. Munsell "chips" (colors and labels) are based on the closest Munsell color found via rgb2Munsell.

Value

a lattice graphics object

Author(s)

D.E. Beaudette

plotMultipleSPC *Plot Multiple SoilProfileCollection Objects*

Description

Plot Multiple SoilProfileCollection Objects

Usage

```
plotMultipleSPC(
  spc.list,
  group.labels,
  args = rep(list(NA), times = length(spc.list)),
  merged.legend = NULL,
  merged.colors = c("#5E4FA2", "#3288BD", "#66C2A5", "#ABDDA4", "#E6F598", "#FEE08B",
    "#FDAE61", "#F46D43", "#D53E4F", "#9E0142"),
  merged.legend.title = merged.legend,
  arrow.offset = 2,
  bracket.base.depth = 95,
  label.offset = 2,
  label.cex = 0.75,
  ...
)
```

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
merged.legend	name of a horizon level attribute from which to create thematic sketches and merged legend
merged.colors	vector of colors used to create thematic sketches from a shared horizon level attribute
merged.legend.title	legend title
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
label.offset	vertical offset of group labels from baseline
label.cex	label size
...	additional arguments to the first call to plotSPC

Details

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

See examples below for usage.

Note

For thematic sketches, use the merged.legend argument instead of color argument to plotSPC

Author(s)

D.E. Beaudette and Ben Marshall

See Also

[profileGroupLabels](#)

Examples

```
##
## Simple Example
##

# using default arguments to plotSPC()

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

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

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

# argument list
arg.list <- list(
  list(name='name', id.style='top'),
  list(name='name', id.style='side')
)

# 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 = arg.list,
  bracket.base.depth = 120, label.cex = 1
)

# specify a different max.depth
plotMultipleSPC(
  spc.list,
  group.labels = c('Collection 1', 'Collection 2'),
  args = arg.list,
  bracket.base.depth = 120, label.cex = 1,
  max.depth = 250
)
```

```
##
## Merged Legend Example
##

# merged legend based on hz attribute 'clay'

# reset sample data
data(sp3)
data(sp4)

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

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

# argument list
arg.list <- list(
  list(name='name', id.style='top'),
  list(name='name', id.style='side')
)

par(mar=c(1,1,3,3))
plotMultipleSPC(
  spc.list,
  group.labels = c('Collection 1', 'Collection 2'),
  args = arg.list,
  label.cex = 1,
  merged.legend = 'clay', merged.legend.title = 'Clay (%)'
)

##
## Complex Merged Legend Example
##

# create a merged legend from "clay" in sp4 and jacobs2000
# use "soil_color" from sp3

# reset sample data
data(sp3)
data(sp4)
data(jacobs2000)

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

# remove 'clay' column from sp3
```

```

sp3$clay <- NULL

# combine into a list
spc.list <- list(sp3, sp4, jacobs2000)

# try some variations on the default arguments
# `clay` is missing in the first SPC, safe to specify another column for colors
arg.list <- list(
  list(color = 'soil_color', id.style='top', name = NA, width = 0.3, hz.depths = TRUE),
  list(name='name', id.style='side', name.style = 'center-center'),
  list(name='name', id.style='side', name.style = 'left-center', hz.depths = TRUE)
)

par(mar=c(1,1,3,3))
plotMultipleSPC(
  spc.list,
  group.labels = c('sp3', 'sp4', 'jacobs2000'),
  label.offset = 3,
  args = arg.list,
  merged.legend = 'clay', merged.legend.title = 'Clay (%)',
  axis.line.offset = 0
)

```

plotSPC

Create Soil Profile Sketches

Description

Generate a diagram of soil profile sketches from a `SoilProfileCollection` object. The [Introduction to SoilProfileCollection Objects tutorial](#) contains many examples and discussion of the large number of arguments to this function.

Usage

```

plotSPC(
  x,
  color = "soil_color",
  width = 0.2,
  name = NULL,
  name.style = "right-center",
  label = idname(x),
  hz.depths = FALSE,
  alt.label = NULL,
  alt.label.col = "black",
  cex.names = 0.5,
  cex.depth.axis = cex.names,
  cex.id = cex.names + (0.2 * cex.names),
  font.id = 2,
  print.id = TRUE,

```

```

    id.style = "auto",
    plot.order = 1:length(x),
    relative.pos = 1:length(x),
    add = FALSE,
    scaling.factor = 1,
    y.offset = rep(0, times = length(x)),
    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,
    shrink.thin = NULL,
    abbr = FALSE,
    abbr.cutoff = 5,
    divide.hz = TRUE,
    hz.distinctness.offset = NULL,
    hz.topography.offset = NULL,
    hz.boundary.lty = NULL,
    axis.line.offset = -2.5,
    plot.depth.axis = TRUE,
    density = NULL,
    show.legend = TRUE,
    col.label = color,
    col.palette = c("#5E4FA2", "#3288BD", "#66C2A5", "#ABDDA4", "#E6F598", "#FEE08B",
                    "#FDAE61", "#F46D43", "#D53E4F", "#9E0142"),
    col.palette.bias = 1,
    col.legend.cex = 1,
    n.legend = 8,
    lwd = 1,
    lty = 1,
    default.color = grey(0.95),
    ...
)

## S4 method for signature 'SoilProfileCollection,ANY'
## note: y argument in generic definition is not currently used
plot(x, y, ...)

```

Arguments

x	a SoilProfileCollection object
color	quoted column name containing R-compatible color descriptions, or numeric / categorical data to be displayed thematically; see details
width	scaling of profile widths (typically 0.1 - 0.4)
name	quoted column name of the (horizon-level) attribute containing horizon designations, can be left as NULL and horizon designation column will be selected via hzdesignname(x). Suppress horizon name printing by setting name=NA.

name.style	one of several possible horizon designations labeling styles: 'right-center' (aqp default), 'left-top', 'left-center'
label	quoted column name of the (site-level) attribute used to identify profile sketches
hz.depths	logical, annotate horizon top depths to the right of each sketch (FALSE)
alt.label	quoted column name of the (site-level) attribute used for secondary annotation
alt.label.col	color used for secondary annotation text
cex.names	baseline character scaling applied to all text labels
cex.depth.axis	character scaling applied to depth scale
cex.id	character scaling applied to label
font.id	font style applied to label, default is 2 (bold)
print.id	logical, print label above/beside each profile? (TRUE)
id.style	label printing style: 'auto' (default) = simple heuristic used to select from: 'top' = centered above each profile, 'side' = 'along the top-left edge of profiles'
plot.order	integer vector describing the order in which individual soil profiles should be plotted
relative.pos	vector of relative positions along the x-axis, within { 1, n}, ignores plot.order see details
add	logical, add to an existing figure
scaling.factor	vertical scaling of profile depths, useful for adding profiles to an existing figure
y.offset	numeric vector of vertical offset for top of profiles in depth units of x, can either be a single numeric value or vector of length = length(x)
x.idx.offset	integer specifying horizontal offset from 0 (left-hand edge)
n	integer describing amount of space along x-axis to allocate, defaults to length(x)
max.depth	suggested lower depth boundary of plot
n.depth.ticks	suggested number of ticks in depth scale
shrink	logical, reduce character scaling for 'long' horizon by 80%
shrink.cutoff	character length defining 'long' horizon names
shrink.thin	integer, horizon thickness threshold for shrinking horizon names by 80%, only activated when shrink = TRUE (NULL = no shrinkage)
abbr	logical, abbreviate label
abbr.cutoff	suggested minimum length for abbreviated label
divide.hz	logical, divide horizons with line segment? (TRUE), see details
hz.distinctness.offset	NULL, or quoted column name (horizon-level attribute) containing vertical offsets used to depict horizon boundary distinctness (same units as profiles), see details and codehzDistinctnessCodeToOffset
hz.topography.offset	NULL, or quoted column name (horizon-level attribute) containing offsets used to depict horizon boundary topography (same units as profiles), see details and codehzTopographyCodeToOffset

<code>hz.boundary.lty</code>	quoted column name (horizon-level attribute) containing line style (integers) used to encode horizon topography
<code>axis.line.offset</code>	horizontal offset applied to depth axis (default is -2.5, larger numbers move the axis to the right)
<code>plot.depth.axis</code>	logical, plot depth axis? (default is TRUE)
<code>density</code>	fill density used for horizon color shading, either a single integer or a quoted column name (horizon-level attribute) containing integer values (default is NULL, no shading)
<code>show.legend</code>	logical, show legend? (default is TRUE)
<code>col.label</code>	thematic legend title
<code>col.palette</code>	color palette used for thematic sketches (default is <code>rev(brewer.pal(10, 'Spectral'))</code>)
<code>col.palette.bias</code>	color ramp bias (skew), see colorRamp
<code>col.legend.cex</code>	scaling of thematic legend
<code>n.legend</code>	approximate number of classes used in numeric legend, max number of items per row in categorical legend
<code>lwd</code>	line width multiplier used for sketches
<code>lty</code>	line style used for sketches
<code>default.color</code>	default horizon fill color used when color attribute is NA
<code>...</code>	other arguments passed into lower level plotting functions
<code>y</code>	(not used)

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.

Adjustments to the legend can be specified via `col.label` (legend title), `col.palette` (palette of colors, automatically expanded), `col.legend.cex` (legend scaling), and `n.legend` (approximate

number of classes for numeric variables, or, maximum number of legend items per row for categorical variables). Currently, plotSPC will only generate two rows of legend items. Consider reducing the number of classes if two rows isn't enough room.

Profile sketches can be added according to relative positions along the x-axis (vs. integer sequence) via `relative.pos` argument. This should be a vector of positions within $\{1, n\}$ that are used for horizontal placement. Default values are `1:length(x)`. Care must be taken when both `plot.order` and `relative.pos` are used simultaneously: `relative.pos` specifies horizontal placement after sorting. `addDiagnosticBracket` and `addVolumeFraction` use the `relative.pos` values for subsequent annotation.

Relative positions that are too close will result in overplotting of sketches. Adjustments to relative positions such that overlap is minimized can be performed with `fixOverlap(pos)`, where `pos` is the original vector of relative positions.

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.

Note

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

Author(s)

D.E. Beaudette

References

Beaudette, D.E., Roudier P., and A.T. O'Geen. 2013. Algorithms for Quantitative Pedology: A Toolkit for Soil Scientists. *Computers & Geosciences*. 52:258 - 268.

See Also

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

Examples

```
# example data
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)

##
## demonstrate horizon designation shrinkage
##

data("jacobs2000")

# shrink "long" horizon names
plotSPC(
  jacobs2000,
  name.style = 'center-center',
  shrink = TRUE,
  cex.names = 0.8
)

# shrink horizon names in "thin" horizons
plotSPC(
  jacobs2000,

```

```

    name.style = 'center-center',
    shrink = TRUE,
    shrink.thin = 15,
    cex.names = 0.8,
  )

##
## demonstrate adaptive legend
##

data(sp3)
depths(sp3) <- id ~ top + bottom

# make some fake categorical data
horizons(sp3)$fake.data <- sample(letters[1:15], size = nrow(sp3), replace=TRUE)

# better margins
par(mar=c(0,0,3,1))

# note that there are enough colors for 15 classes (vs. previous limit of 10)
# note that the legend is split into 2 rows when length(classes) > n.legend argument
plot(sp3, color='fake.data', name='fake.data', cex.names=0.8)

# make enough room in a single legend row
plot(sp3, color='fake.data', name='fake.data', cex.names=0.8, n.legend=15)

##
## demonstrate y.offset argument
## must be of length 1 or length(x)
##

# example data and local copy
data("jacobs2000")
x <- jacobs2000

# y-axis offsets, simulating a elevation along a hillslope sequence
# same units as horizon depths in `x`
y.offset <- c(-5, -10, 22, 65, 35, 15, 12)

par(mar = c(0, 0, 2, 2))

# y-offset at 0
plotSPC(x, color = 'matrix_color', cex.names = 0.66)

# constant adjustment to y-offset
plotSPC(x, color = 'matrix_color', cex.names = 0.66, y.offset = 50)

# attempt using invalid y.offset
# warning issued and default value of '0' used
# plotSPC(x, color = 'matrix_color', cex.names = 0.66, y.offset = 1:2)

```

```

# variable y-offset
par(mar = c(0, 0, 2, 0))
plotSPC(
  x,
  y.offset = y.offset,
  color = 'matrix_color',
  cex.names = 0.66,
  hz.depths = TRUE,
  name.style = 'center-center'
)

# random y-axis offsets
yoff <- runif(n = length(x), min = 1, max = 100)
# random gradient of x-positions
xoff <- runif(n = length(x), min = 1, max = length(x))

# align / adjust relative x positions
set.seed(111)
pos <- alignTransect(xoff, x.min = 1, x.max = length(x))

par(mar = c(0.5, 0.5, 0.5, 0.5))
plotSPC(x,
  plot.order = pos$order,
  relative.pos = pos$relative.pos,
  y.offset = y.offset,
  color = 'matrix_color',
  cex.names = 0.66,
  hz.depths = TRUE,
  name.style = 'center-center'
)

box()

```

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)
```

pms.munsell.lut

Pantone Colors / Munsell Lookup Table

Description

A simple lookup table to convert **Pantone spot colors** into Munsell notation. Association is based on the "closest" Munsell color via **CIE2000 distance metric (dE00)**. This is an experimental association between the two color systems and should not be used for precision color matching or mixing applications.

Possible uses include rough estimation of soil colors in the field, by means of color swatches based on the Pantone system. This type of color matching is most appropriate in an educational setting where official soil color books may be too expensive.

Usage

```
data(pms.munsell.lut)
```

Format

code Pantone spot color code

hex hex representation of sRGB colorspace, suitable for on-screen use

munsell Munsell notation of closest color "chip"

dE00 delta-E 2000 metric describing the (perceptual) distance to the closest Munsell chip

Details

For now, lookup is performed by manual subset (see examples 1 and 2 below) or implicit subset by way of a join (example 3). Reverse lookup (Munsell -> Pantone) will not always result in a matching color, see example 3 below.

Note

The lookup table contains entries for both coated and un-coated colors, these are identified by a '-c' or '-u' suffix. For example, Pantone code '100-c' is associated with '10Y 9/9'.

Several Munsell chips are matched by multiple Pantone spot colors, e.g. 5YR 5/5.

```
1 2 3 4 5 6 8 9 0.65 0.24 0.08 0.02 0.01 0.00 0.00 0.00
```

References

Data were sourced from:

- coated colors: <https://raw.githubusercontent.com/ajesma/Pantoner/gh-pages/csv/pantone-coated.csv>
- uncoated colors: <https://github.com/ajesma/Pantoner/raw/gh-pages/csv/pantone-uncoated.csv>

Examples

```
# load LUT
data(pms.munsell.lut)

## 1. Pantone -> Munsell

# colors to match
colors <- c('10YR 3/3', '7.5YR 4/6')

# index / subset match
idx <- pms.munsell.lut$munsell %in% colors
m <- pms.munsell.lut[idx, ]

# simple display
colorContrastPlot(m1 = m$munsell[1], m2 = m$munsell[2], labels = m$code)
```

```

## 2. Munsell -> Pantone
colors <- c('723-c', '451-c')

# index / subset match
idx <- pms.munsell.lut$code %in% colors
m <- pms.munsell.lut[idx, ]

# simple display
colorContrastPlot(m1 = m$munsell[1], m2 = m$munsell[2], labels = m$code)
## 3. integration with SPC
data(pms.munsell.lut)
data(sp6)
depths(sp6) <- id ~ top + bottom

# get the closest Munsell chip from color meter data
sp6$munsell <- getClosestMunsellChip(sp6$color, convertColors = FALSE)

# prepare an intermediate data.frame for performing join to LUT
h <- horizons(sp6)[, c(hzidname(sp6), 'munsell')]

# left join
# not all Munsell colors have a paired Pantone color
m <- merge(h, pms.munsell.lut, by = 'munsell', all.x = TRUE, sort = FALSE)

# splice into original SPC
horizons(sp6) <- m

# graphical check
par(mar = c(0, 0, 2, 1))
plotSPC(sp6, color = 'hex')

## 4. multiple Pantone colors matching a single Munsell color
#
colors <- pms.munsell.lut[pms.munsell.lut$munsell == '5YR 5/5', ]
colors <- colors[order(colors$dE00), ]

par(mar = c(0, 0, 2, 0), fg = 'white', bg = 'black')
soilPalette(colors$hex, lab = colors$code)
title('Pantone Colors Roughly Matching 5YR 5/5', col.main = 'white', line = 0)

```

```
previewColors
```

```
Preview Colors
```

Description

Preview colors arranged according to CIE2000 distances or manual specification.

Usage

```
previewColors(  
  cols,  
  method = c("grid", "MDS", "manual"),  
  col.order = NULL,  
  nrow = ceiling(sqrt(length(cols))),  
  ncol = nrow,  
  border.col = "black",  
  pt.cex = 2,  
  pt.pch = 15  
)
```

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
pt.pch	point symbol 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 `MASS::cmdscale`.

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

Value

When `method = "grid"` or `"manual"` a vector of color order is returned. When `method = "MDS"`, the output from `MASS::cmdscale`.

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)

# create colors using HCL space
cols.hcl <- hcl(h = 0:360, c = 100, l = 50)

# grid, colors sorted by dE00
previewColors(cols.hcl)

# manual specification
previewColors(cols.hcl, method = 'manual', col.order = 1:361)

# MDS
previewColors(cols.hcl, method = 'MDS', pt.cex = 1)

```

profileApply

Iterate over profiles in a SoilProfileCollection

Description

Iterate over all profiles in a SoilProfileCollection, calling FUN on a single-profile SoilProfileCollection for each step.

Usage

```

## S4 method for signature 'SoilProfileCollection'
profileApply(
  object,
  FUN,
  simplify = TRUE,
  frameify = FALSE,
  chunk.size = 100,
  column.names = NULL,
  ...
)

```

Arguments

object a SoilProfileCollection

<code>FUN</code>	a function to be applied to each profile within the collection
<code>simplify</code>	logical, should the result be simplified to a vector? default: TRUE; see examples
<code>frameify</code>	logical, should the result be collapsed into a data.frame? default: FALSE; overrides <code>simplify</code> argument; see examples
<code>chunk.size</code>	numeric, size of "chunks" for faster processing of large <code>SoilProfileCollection</code> objects; default: 100
<code>column.names</code>	character, optional character vector to replace <code>frameify</code> -derived column names; should match length of <code>colnames()</code> from <code>FUN</code> result; default: NULL
<code>...</code>	additional arguments passed to <code>FUN</code>

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. When `frameify` is TRUE, a data.frame is returned. An attempt is made to identify `idname` and/or `hzidname` in the data.frame result, safely ensuring that IDs are preserved to facilitate merging `profileApply` result downstream.

Examples

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

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

# scale a single property 'prop' in horizon table
# 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, calculate on each horizon, merge into original SPC

# example data
data(sp1)

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

# calculate horizon thickness and proportional thickness
# returns a data.frame result with multiple attributes per horizon
thicknessFunction <- function(p) {
  hz <- horizons(p)
  depthnames <- horizonDepths(p)
  res <- data.frame(profile_id(p), hzID(p),
                    thk=(hz[[depthnames[[2]]]] - hz[[depthnames[1]]]))
  res$hz_prop <- res$thk / sum(res$thk)
  colnames(res) <- c(idname(p), hzidname(p), 'hz_thickness', 'hz_prop')
  return(res)
}

# list output option with simplify=F, list names are profile_id(sp1)
list.output <- profileApply(sp1, thicknessFunction, simplify = FALSE)
head(list.output)

# data.frame output option with frameify=TRUE
df.output <- profileApply(sp1, thicknessFunction, frameify = TRUE)
head(df.output)

# since df.output contains idname(sp1) and hzidname(sp1),
# it can safely be merged by a left-join via horizons<- setter
horizons(sp1) <- df.output

plot(density(sp1$hz_thickness, na.rm=TRUE), main="Density plot of Horizon Thickness")

## 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$dep <- profileApply(sp1, function(i) {round(rnorm(n=1, mean=mean(i$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
slot(i, 'horizons') <- 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 <- pbindlist(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)

```

profileGroupLabels *Soil Profile Group Labels*

Description

Labels groups of soil profiles within soil profile sketches.
See examples below for ideas.

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

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_id<-          Set profile IDs
```

Description

Set vector containing profile IDs

Get or set a vector of profile IDs

Arguments

value	a unique vector of equal length to number of profiles length(object)
object	a SoilProfileCollection

```
proj4string,SoilProfileCollection-method
          Set PROJ4 string for the SoilProfileCollection
```

Description

Set PROJ4 string for the SoilProfileCollection

Usage

```
## S4 method for signature 'SoilProfileCollection'
proj4string(obj)
```

Arguments

obj	A SoilProfileCollection
-----	-------------------------

```
proj4string<- ,SoilProfileCollection,ANY-method
          Set PROJ4 string for the SoilProfileCollection
```

Description

Set PROJ4 string for the SoilProfileCollection

Usage

```
## S4 replacement method for signature 'SoilProfileCollection,ANY'
proj4string(obj) <- value
```

Arguments

obj	A SoilProfileCollection
value	A proj4string

random_profile	<i>Random Profile</i>
----------------	-----------------------

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,
  SPC = 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
SPC	result is a SoilProfileCollection object, otherwise a data.frame object
...	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 (https://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 data.frame or SoilProfileCollection object.

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, result is a list of SoilProfileCollection objects
d <- lapply(1:10, random_profile, SPC=TRUE)

# combine
d <- combine(d)

# plot
opar <- par(mar=c(0,0,3,2))
plotSPC(d, color='p1', name='name', cex.names=0.75)
par(opar)
```

```

# simulate horizon boundary distinctness codes:
d <- lapply(1:10, random_profile, SPC=TRUE, HzDistinctSim=TRUE)
d <- combine(d)

d$HzD <- hzDistinctnessCodeToOffset(d$HzDistinctCode)

opar <- par(mar=c(0,0,3,2))
plotSPC(d, name='name', color='p1', hz.distinctness.offset='HzD')
par(opar)

# depth functions are generated using the LPP function
opar <- par(mfrow=c(2,1), mar=c(0,0,3,0))

# generate data
d.1 <- lapply(1:10, random_profile, SPC=TRUE, n=c(6, 7, 8), n_prop=1, method='LPP')
d.1 <- combine(d.1)

# plot
plotSPC(d.1, name='name', color='p1', col.label = 'LPP Defaults')

# do this again, this time set all of the LPP parameters
d.2 <- lapply(1:10, random_profile, SPC=TRUE, 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)
d.2 <- combine(d.2)

# plot
plotSPC(d.2, name='name', color='p1', col.label = 'Custom LPP Parameters')

# reset plotting defaults
par(opar)

# try plotting the LPP-derived simulated data
# aggregated over all profiles
a <- slab(d.2, 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.2)
}

```



```
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*Rebuild a SoilProfileCollection object*

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.

A valid SoilProfileCollection object.

Author(s)

D.E. Beaudette

D.E. Beaudette, A.G. Brown

See Also

[checkSPC](#) Rebuild a SoilProfileCollection object

Rebuild a SoilProfileCollection object

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

[checkSPC](#)

reorderHorizons	<i>Re-order corrupted horizon data</i>
-----------------	--

Description

This is a method made available primarily to repair horizon data that have been corrupted relative to their order at time of SoilProfileCollection construction.

There is an option to specify the target order, but this will not update the corresponding metadata entry tracking the original order. Use this functionality at your own risk.

Usage

```
## S4 method for signature 'SoilProfileCollection'
reorderHorizons(object, target.order = NULL)
```

Arguments

object	A SoilProfileCollection
target.order	A numeric vector of equal length to object. Default value is NULL which restores the internal order of the collection.

Value

SoilProfileCollection

repairMissingHzDepths	<i>Repair Problematic Lower Horizon Depths</i>
-----------------------	--

Description

Attempt a simple repair of horizon bottom depths in the presence of NA, or in cases where the horizon shares a common top and bottom depth. Both situations are common in pedon description where "contact" (Cd, Cr, R, etc.) was described without a lower depth. This repair is only applied to the deepest horizon within a profile as identified by [getLastHorizonID](#).

Usage

```
repairMissingHzDepths(x, adj = 10)
```

Arguments

x	SoilProfileCollection
adj	vertical offset applied to "repair" affected bottom depths

Value

SoilProfileCollection with a new (logical) horizon-level attribute `.repaired` marking affected horizons

replaceHorizons<- *Replace data in the horizon slot*

Description

Replaces horizon data with new `data.frame` object.

Arguments

object	A SoilProfileCollection
value	An object inheriting <code>data.frame</code>

Examples

```
# load test data
data(sp2)

# promote to SPC
depths(sp2) <- id ~ top + bottom

# one profile
p <- sp2[1,]

# 23 variables in horizon data
length(horizonNames(sp2))

# remove all but essential ones
replaceHorizons(p) <- horizons(p)[,c(idname(p),hzidname(p),horizonDepths(p))]

# inspect result (a clean slate)
horizons(p)
```

resample.twotheta *Resample an XRD Pattern*

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')
```

restrictions	<i>Retrieve restriction data from SoilProfileCollection</i>
--------------	---

Description

Get restriction data from SoilProfileCollection. Result is returned in the same data.frame class used to initially construct the SoilProfileCollection.

Arguments

object	a SoilProfileCollection
--------	-------------------------

restrictions<-	<i>Add data to the restrictions slot</i>
----------------	--

Description

Restrictions data in an object inheriting from data.frame can easily be added via merge (LEFT JOIN). There must be one or more same-named profile ID columns on the left and right hand side to facilitate the join: restrictions(spc) <-newdata.

Arguments

object	A SoilProfileCollection
value	An object inheriting data.frame

Examples

```
# load test data
data(sp2)

# promote to SPC
depths(sp2) <- id ~ top + bottom

# assign abrupt textural change to a profile
newdata <- data.frame(id = c("hon-21"),
                      restrkind = "abrupt textural change",
                      restrdep = 46)
```

```
# do left join
restrictions(sp2) <- newdata

# inspect site table: newvalue TRUE only for horizons
# with top depth equal to zero
restrictions(sp2)
```

 rgb2munsell

sRGB to Munsell Color Conversion

Description

Convert sRGB color coordinates to the closest n Munsell chips in the munsell lookup table.

Usage

```
rgb2munsell(color, colorSpace = c("CIE2000", "LAB", "sRGB"), nClosest = 1)
```

Arguments

color	a data.frame or matrix object containing sRGB coordinates in the range of (0,1)
colorSpace	distance metric (colorspace) to use for finding the closest chip: CIE2000 is the most accurate but requires farver $\geq 2.0.3$, Euclidean distance in CIELAB is a close second, while Euclidean distance in sRGB is not at all accurate and should only be used for demonstration purposes.
nClosest	number of closest Munsell colors to return (valid range is 1-20)

Value

an (NA-padded) data.frame containing hue, value, chroma, and distance (dE00 when colorSpace = 'CIE2000', Euclidean distance otherwise) to nearest matching color.

Note

This function is fully vectorized and will pad output with NA-records when NA are present in color.

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> <https://www.munsellcolourscienceforpainters.com/MunsellAndKubelkaMunkToolbox/MunsellAndKubelkaMunkToolbox.html>

Examples

```
# Munsell notation to sRGB triplets [0-1]
color <- munsell2rgb(
  the_hue = c('10YR', '2.5YR'),
  the_value = c(3, 5),
  the_chroma = c(5, 6),
  return_triplets = TRUE
)

# result is a data.frame
color

# sRGB triplets to closest Munsell color
# dE00 distance metric
# result is a data.frame
rgb2munsell(color)
```

ROSETTA.centroids	<i>Average Hydraulic Parameters from the ROSETTA Model by USDA Soil Texture Class</i>
-------------------	---

Description

Average soil hydraulic parameters generated by the first stage predictions of the ROSETTA model by USDA soil texture class. These data were extracted from [ROSETTA documentation](#) and reformatted for ease of use.

Usage

```
data(ROSETTA.centroids)
```

Format

A data frame:

texture soil texture class, ordered from low to high available water holding capacity

theta_r average saturated water content

theta_s average residual water content

alpha average value, related to the inverse of the air entry suction, log10-transformed values with units of cm

npar average value, index of pore size distribution, log10-transformed values with units of 1/cm

theta_r_sd 1 standard deviation of theta_r

theta_s_sd 1 standard deviation of theta_s

alpha_sd 1 standard deviation of alpha

- npar_sd** 1 standard deviation of npar
- sat** approximate volumetric water content at which soil material is saturated
- fc** approximate volumetric water content at which matrix potential = -33kPa
- pwp** approximate volumetric water content at which matrix potential = -1500kPa
- awc** approximate available water holding capacity: VWC(-33kPa)
 - VWC(-1500kPa)

Details

Theoretical water-retention parameters for uniform soil material of each texture class have been estimated via van Genuchten model.

See [the related tutorial](#)

Source

[ROSETTA Class Average Hydraulic Parameters](#)

References

van Genuchten, M.Th. (1980). "A closed-form equation for predicting the hydraulic conductivity of unsaturated soils". Soil Science Society of America Journal. 44 (5): 892-898.

Schaap, M.G., F.J. Leij, and M.Th. van Genuchten. 2001. ROSETTA: A computer program for estimating soil hydraulic parameters with hierarchical pedotransfer functions. Journal of Hydrology 251(3-4): 163-176.

Examples

```
## Not run:

library(aqp)
library(soilDB)
library(latticeExtra)

data("ROSETTA.centroids")

# iterate over horizons and generate VG model curve
res <- lapply(1:nrow(ROSETTA.centroids), function(i) {
  m <- KSSL_VG_model(VG_params = ROSETTA.centroids[i, ], phi_min = 10^-3, phi_max=10^6)$VG_curve
  # copy generalized hz label
  m$hz <- ROSETTA.centroids$hz[i]
  # copy ID
  m$texture_class <- ROSETTA.centroids$texture[i]
  return(m)
})

# copy over lab sample number as ID
res <- do.call('rbind', res)
```



```
# check: OK
str(res)

# visual check: OK
xyplot(
  phi ~ theta | texture_class, data=res,
  type=c('l', 'g'),
  scales=list(alternating=3, x=list(tick.number=10), y=list(log=10, tick.number=10)),
  yscale.components=yscale.components.logpower,
  ylab=expression(Suction~(kPa)),
  xlab=expression(Volumetric~Water~Content~(cm^3/cm^3)),
  par.settings = list(superpose.line=list(col='RoyalBlue', lwd=2)),
  strip=strip.custom(bg=grey(0.85)),
  as.table=TRUE
)

## End(Not run)
```

rowley2019

Soil Morphologic, Geochemical, and Mineralogy Data from Rowley et al. 2019.

Description

Data from Table 1 and Supplementary Tables 1 and 2 from "A cascading influence of calcium carbonate on the biogeochemistry and pedogenic trajectories of subalpine soils, Switzerland".

Usage

```
data(rowley2019)
```

Format

A SoilProfileCollection object:

site-level attributes

id profile ID

group profile group

horizon-level attributes

sample_id sample ID

name horizon name

pH pH
Al_exch cmol(+) / kg, exchangeable Al
Ca_exch cmol(+) / kg, exchangeable Ca
CEC_sum cmol(+) / kg, cation exchange capacity calculated as the sum of exchangeable cations, not including H⁺
Ca_exch_saturation percent
Al_exch_saturation percent
TON percent, total nitrogen
SOC percent, soil organic carbon
C_to_N carbon to nitrogen ratio
Al_o g/kg, oxalate-extractable Al
Fe_o g/kg, oxalate-extractable Fe
Al_d g/kg, dithionite-extractable Al
Fe_d g/kg, dithionite-extractable Fe
Fe_o_Fe_d Fe_o to Fe_d ratio
id profile ID
top horizon top (cm)
bottom horizon bottom (cm)
Al g/kg by x-ray fluorescence
Ca g/kg by x-ray fluorescence
Cr g/kg by x-ray fluorescence
Fe g/kg by x-ray fluorescence
K g/kg by x-ray fluorescence
Mg g/kg by x-ray fluorescence
Mn g/kg by x-ray fluorescence
Na g/kg by x-ray fluorescence
Ni g/kg by x-ray fluorescence
P g/kg by x-ray fluorescence
Si g/kg by x-ray fluorescence
Ti g/kg by x-ray fluorescence
Phyllosilicates percent by x-ray diffraction spectra
Quartz percent by x-ray diffraction spectra
K_Feldspar percent by x-ray diffraction spectra
Na_Plagioclase percent by x-ray diffraction spectra
Goethite percent by x-ray diffraction spectra
Unidentified percent by x-ray diffraction spectra
CCE_Total percent

CCE_Reactive percent
Reactive_carbonate percent
Sand percent <2um
Silt percent 2-50um
Clay percent 50-2000um
CaH2O Milliq ex: grams of Ca per kilogram of dry soil (g kg-1)
Ca2MKCl 2M KCl: grams of Ca per kilogram of dry soil (g kg-1)
CaNa2EDTA 0.05 M Na2EDTA: grams of Ca per kilogram of dry soil (g kg-1)
CaCuCl2 0.5 M CuCl2: grams of Ca per kilogram of dry soil (g kg-1)
hzID horizon ID

References

Mike C. Rowley, Stephanie Grand, Thierry Adatte, Eric P. Verrecchia, Cascading influence of calcium carbonate on the biogeochemistry and pedogenic trajectories of subalpine soils), Switzerland, *Geoderma*, 2019, 114065, ISSN 0016-7061, doi: [10.1016/j.geoderma.2019.114065](https://doi.org/10.1016/j.geoderma.2019.114065).

Examples

```

library(lattice)

# load data
data('rowley2019')

# check first 5 rows and 10 columns of horizon data
horizons(rowley2019)[1:5, 1:10]

# check site data
site(rowley2019)

# graphical summary
par(mar=c(1,1,3,1))
plotSPC(rowley2019, color='Feo_Fed', name='name', cex.names=0.85)

plotSPC(rowley2019, color='Ca_exch', name='name', cex.names=0.85)

# grouped plot
groupedProfilePlot(rowley2019, groups = 'group', color='Ca_exch',
name='name', cex.names=0.85, group.name.offset = -10)

# aggregate over 1cm slices, for select properties
a <- slab(rowley2019, group ~ Reactive_carbonate + Ca_exch + pH + K_Feldspar + Na_Plagioclase + Al)

# plot styling
tps <- list(superpose.line=list(lwd=2, col=c('royalblue', 'firebrick')))

# make the figure
xyplot(top ~ p.q50 | variable, data=a, ylab='Depth', groups=group,

```

```
main='', as.table=TRUE,
xlab='median bounded by 25th and 75th percentiles',
lower=a$p.q25, upper=a$p.q75, ylim=c(55,-5),
panel=panel.depth_function,
prepanel=prepanel.depth_function,
cf=a$contributing_fraction,
alpha=0.33, sync.colors=TRUE,
scales=list(x=list(relation='free', alternating=1)),
par.settings=tps,
auto.key=list(columns=2, lines=TRUE, points=FALSE),
strip=strip.custom(bg=grey(0.9))
)
```

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)
```

segment

*Segmenting of Soil Horizon Data by Depth Interval***Description**

This function adds depth interval ("segment") labels to soil horizon data associated with `SoilProfileCollection` and `data.frame` objects. Additional horizon records are inserted when a segment label does not overlap with a horizon boundary. See examples.

Usage

```
segment(object, intervals, trim = TRUE, hzdepcols = NULL)
```

Arguments

<code>object</code>	either a <code>SoilProfileCollection</code> or <code>data.frame</code>
<code>intervals</code>	a vector of integers over which to slice the horizon data (e.g. <code>c(25,100)</code> or <code>25:100</code>)
<code>trim</code>	logical, when <code>TRUE</code> horizons in <code>object</code> are truncated to the min/max specified in <code>intervals</code> . When <code>FALSE</code> , those horizons overlapping an interval are marked as such. Care should be taken when specifying more than one depth interval and <code>trim = FALSE</code> .
<code>hzdepcols</code>	a character vector of length 2 specifying the names of the horizon depths (e.g. <code>c("hzdept", "hzdepb")</code>), only necessary if <code>object</code> is a <code>data.frame</code> .

Details

This function adds segment labels to soil horizon data according to `intgervals` (e.g. `c(25, 100)` or `25:100`). Compared to `slice`, `slab`, and `glom`, `segment` performs no aggregation or resampling of the source data, rather, labels are added to horizon records for subsequent aggregation. This makes it possible to process a very large number of records outside of the constraints associated with e.g. `slice` or `slab`.

Value

Either a `SoilProfileCollection` or `data.frame` with the original horizon data segmented by depth intervals. There are usually more records in the resulting object, one for each time a segment interval partially overlaps with a horizon. A new column called `segment_id` identifying the depth interval is added.

Author(s)

Stephen Roecker

Examples

```
# example data
data(sp1)

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

# segment and trim
z <- segment(sp1, intervals = c(0, 10, 20, 30), trim = TRUE)

# display segment labels
# note that there are new horizon boundaries at segments
par(mar = c(0, 0, 3, 1))
plotSPC(z, color = 'segment_id', width = 0.3)

# highlight new horizon records
par(mar = c(0, 0, 2, 1))
plotSPC(z, color = NA, default.color = NA, width = 0.3, lwd = 1)
plotSPC(sp1, color = NA, default.color = NA,
width = 0.3, lwd = 3, add = TRUE, name = NA, print.id = FALSE)
legend('top', horiz = TRUE,
legend = c('original', 'segmented'),
lwd = c(1, 3), cex = 0.85, bty = 'n')

# same results as slab()
# 10 random profiles
s <- lapply(1:10, random_profile, n_prop = 1, SPC = TRUE, method = 'random_walk')
s <- combine(s)

a.slab <- slab(s, fm = ~ p1, slab.structure = c(0, 10, 20, 30), slab.fun = mean, na.rm = TRUE)

z <- segment(s, intervals = c(0, 10, 20, 30), trim = TRUE)
z <- horizons(z)
z$thick <- z$bottom - z$top

a.segment <- sapply(split(z, z$segment_id), function(i) {
  weighted.mean(i$p1, i$thick)
})

res <- data.frame(
  slab = a.slab$value,
  segment = a.segment,
  diff = a.slab$value - a.segment
)
```

```
print(res)
res$diff < 0.001

data(sp5)

# segment by upper 25-cm
test1 <- segment(sp5, intervals = c(0, 100))
print(test1)
nrow(test1)
print(object.size(test1), units = "Mb")

# segment by 1-cm increments
test2 <- segment(sp5, intervals = 0:100)
print(test2)
nrow(test2)
print(object.size(test2), units = "Mb")

# segment and aggregate
test3 <- segment(horizons(sp5),
                 intervals = c(0, 5, 15, 30, 60, 100, 200),
                 hzdepcols = c("top", "bottom")
)
test3$hzthk <- test3$bottom - test3$top
test3_agg <- by(test3, test3$segment_id, function(x) {
  data.frame(
    hzID = x$hzID[1],
    segment_id = x$segment_id[1],
    average = weighted.mean(x$clay, w = x$hzthk)
  )
})
test3_agg <- do.call("rbind", test3_agg)

head(test3_agg)
```

shannonEntropy

Shannon Entropy

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

When `b = length(x)` the result is the normalized Shannon entropy of (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)
```

show

SoilProfileCollection show method

Description

Pretty output method for `SoilProfileCollection` objects. By default this method limits output to 10 columns and 6 rows from the site and horizon tables respectively.

There is an `aqp` environment option you can set to increase the number of columns shown by default: `options(.aqp.show.n.cols = 100)`,

Usage

```
## S4 method for signature 'SoilProfileCollection'  
show(object)
```

Arguments

object a SoilProfileCollection

Examples

```
# load a SoilProfileCollection  
data(sp5)  
  
# use the show() method  
show(sp5)  
  
# which is same as this (in the console)  
sp5
```

sierraTransect	<i>Soil Physical and Chemical Data Related to Studies in the Sierra Nevada Mountains, CA, USA.</i>
----------------	--

Description

Soil physical and chemical data associated with two bio-climatic sequences (granitic and andesitic parent material) from the western flank of the Sierra Nevada mountains.

Usage

```
data(sierraTransect)
```

Format

An object of class SoilProfileCollection of length 14.

Details

These data were assembled from Dahlgren et al. (1997) and Rasmussen et al. (2007), with permission granted by lead authors, by D.E. Beaudette.

Source

Original manuscripts and person communication with authors.

References

R.A. Dahlgren, J.L. Boettinger, G.L. Huntington, R.G. Amundson. Soil development along an elevational transect in the western Sierra Nevada, California, *Geoderma*, Volume 78, Issues 3–4, 1997, Pages 207-236.

Rasmussen, C., Matsuyama, N., Dahlgren, R.A., Southard, R.J. and Brauer, N. (2007), Soil Genesis and Mineral Transformation Across an Environmental Gradient on Andesitic Lahar. *Soil Sci. Soc. Am. J.*, 71: 225-237.

Examples

```
data(sierraTransect)

# tighter margins
op <- par(mar=c(0,0,0,0))

# quick sketch
plotSPC(sierraTransect, name.style = 'center-center', width=0.3)

# split by transect
par(mar=c(0,0,1,1))
groupedProfilePlot(
  sierraTransect, groups='transect',
  group.name.offset = -15, width=0.3,
  name.style='center-center'
)

# thematic
groupedProfilePlot(
  sierraTransect, groups='transect',
  group.name.offset = -15, width=0.3,
  name.style='center-center', color='Fe_o_to_Fe_d'
)

# horizon boundary viz
sierraTransect$hzd <- hzDistinctnessCodeToOffset(substr(sierraTransect$hz_boundary, 0, 1))

groupedProfilePlot(
  sierraTransect, groups='transect', group.name.offset = -15,
  width=0.3, name.style='center-center', color='Fe_o_to_Fe_d',
  hz.distinctness.offset='hzd')

# split transects
g <- subset(sierraTransect, transect == 'Granite')
a <- subset(sierraTransect, transect == 'Andesite')

g.order <- order(g$elev)
a.order <- order(a$elev)

# order (left -> right) by elevation
```

```

par(mar=c(2,0,0,2), mfrow=c(2,1))
plot(g, width=0.3, name.style='center-center', cex.names=0.75, plot.order=g.order)
axis(1, at=1:length(g), labels=g$elev[g.order], line=-1.5)

plot(a, width=0.3, name.style='center-center', cex.names=0.75, plot.order=a.order)
axis(1, at=1:length(a), labels=a$elev[a.order], line=-1.5)

par(op)

```

sim

DEPRECATED Simulate Soil Profiles

Description

Simulate a collection of soil profiles based on the horization of a single soil profile. Now deprecated: use [perturb\(\)](#) for perturbations of horizon thicknesses or boundaries.

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	minimum 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 perturb](#)

Examples

```
# please see documentation for perturb() for examples
# the sim() function calls perturb() internally
```

site	<i>Retrieve site data from SoilProfileCollection</i>
------	--

Description

Get site data from SoilProfileCollection. Result is returned in the same `data.frame` class used to initially construct the SoilProfileCollection.

There are two options available via the `site<-` setter.

The first is a "normalization" by formula interface, whereby one specifies an attribute that is constant in horizons within profiles to be promoted to a site-level variable: `site(spc) <- ~ horizonvariable`

The second is creation of site data from an external `data.frame` via merge (LEFT JOIN). There must be one or more same-named columns (with at least some matching data) on the left and right hand side to facilitate the join: `site(spc) <- newdata`

Usage

```
## S4 method for signature 'SoilProfileCollection'
site(object)

site(object) <- value
```

Arguments

object	A SoilProfileCollection
value	A formula or object inheriting <code>data.frame</code>

Examples

```
# load test data
data(sp2)

# promote to SPC
depths(sp2) <- id ~ top + bottom

# normalize a horizon-level attribute to site
site(sp2) <- ~ surface
```

```

# inspect site table
site(sp2)

# make some data: classify two geomorphic surfaces with numeric value
newdata <- data.frame(surface = c("holocene",
                                "lower riverbank"),
                      newvalue = c(1,2))

# do left join based on newly-normalized "surface" attribute
site(sp2) <- newdata

# inspect site table: holocene & lower riverbank have values
site(sp2)

```

```

siteNames<-          Set site column names

```

Description

Set site column names
 Get names of columns in site table.

Arguments

value	a unique vector of equal length to number of columns in site: <code>length(siteNames(object))</code>
object	a <code>SoilProfileCollection</code>

```

slab-methods        Slab-Wise Aggregation of SoilProfileCollection Objects

```

Description

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

Usage

```

## S4 method for signature 'SoilProfileCollection'
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 <code>groups ~ var1 + var2 + var3'</code> where named variables are aggregated within groups' OR where named variables are aggregated across the entire collection <code>~ var1 + var2 + var3'</code> . If <code>groups'</code> 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>stats::quantile</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 `stats::quantile` 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.

The default `slab.fun` was changed 2019-10-30 from a wrapper around `Hmisc::hdquantile` to a wrapper around `stats::quantile`. See examples below for a simple way to switch to the HD quantile estimator.

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

list()

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](#), [quantile](#)**Examples**

```
##
## basic examples
##
library(lattice)
library(grid)
library(data.table)

# 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,
alpha=0.5,
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
##

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

# 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)

# convert wide -> long for plotting
# result is a data.table
# genhz factor levels are set by order in `measure.vars`
a.long <- melt(
  as.data.table(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,
       col = 1, lwd = 2,
       xlab = 'Class Probability',
       ylab = 'Depth (cm)',
       strip = strip.custom(bg = grey(0.85)),
       scales = list(x = list(alternating = FALSE)),
       ylim = c(150, -5), type=c('S','g'),
       horizontal = TRUE, layout = c(4,1)
       )

# 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,
       lwd = 2,
       auto.key = list(
         lines = TRUE,
         points = FALSE,
         cex = 0.8,
         columns = 1,
         space = 'right'
       )
)

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

# convert wide -> long for plotting
# result is a data.table
# genhz factor levels are set by order in `measure.vars`
a.1.long <- melt(
  as.data.table(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), lwd = 2)),
    scales = list(alternating = 3),
    xlab = 'Class Probability',
    ylab = 'Depth (cm)',
    strip = strip.custom(bg = grey(0.85))
)

# 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:
##
## HD quantile estimator
##

library(soilDB)
library(lattice)
library(data.table)

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

# default slab.fun wraps stats::quantile()
a <- slab(loafercreek, fm = ~ total_frgs_pct + clay)

# use HD quantile estimator from Hmisc package instead
a.HD <- slab(loafercreek, fm = ~ total_frgs_pct + clay, slab.fun = aqp::slab.fun.numeric.HD)

# combine
g <- make.groups(standard=a, HD=a.HD)

# note differences
densityplot(~ p.q50 | variable, data=g, groups=which,
            scales=list(relation='free', alternating=3, tick.number=10, y=list(rot=0)),
            xlab='50th Percentile', pch=NA, main='Loafercreek',
            auto.key=list(columns=2, points=FALSE, lines=TRUE),
            par.settings=list(superpose.line=list(lwd=2, col=c('RoyalBlue', 'Orange2'))))
)

# differences are slight but important
xyplot(
  top ~ p.q50 | variable, data=g, groups=which,
  xlab='Value', ylab='Depth (cm)',
  asp=1.5, main='Loafercreek',
  lower=g$p.q25, upper=g$p.q75,
  sync.colors=TRUE, alpha=0.25, cf=g$contributing_fraction,
  ylim=c(115,-5), layout=c(2,1), scales=list(x=list(relation='free')),
  par.settings=list(superpose.line=list(lwd=2, col=c('RoyalBlue', 'Orange2'))),

```

```

strip=strip.custom(bg=grey(0.85)),
panel=panel.depth_function,
prepanel=prepanel.depth_function,
auto.key=list(columns=2, lines=TRUE, points=FALSE)
)

##
## 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
xyplot(
  top ~ mean | variable, data=a, groups=group,
  lower=a$lower, upper=a$upper,
  sync.colors=TRUE, alpha=0.5,
  cf = a$contributing_fraction,
  xlab = 'Mean Bounded by +/- 1SD',
  ylab = 'Depth (cm)',
  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,
  strip = strip.custom(bg=grey(0.85)),
  auto.key = list(columns=2, lines=TRUE, points=FALSE)
)

# compare a single profile to the group-level aggregate values

```

```

a.1 <- slab(sp3[, ], 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, subscripts=TRUE,
  lower=a$lower, upper=a$upper, ylim=c(125,-5),
  layout=c(3,1), scales=list(x=list(relation='free')),
  xlab = 'Mean Bounded by +/- 1SD',
  ylab = 'Depth (cm)',
  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)
    )
  ),
  strip = strip.custom(bg=grey(0.85)),
  auto.key = list(columns=3, lines=TRUE, points=FALSE)
)

## 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
# convert long -> wide
dcast(
  as.data.table(a),
  formula = group + top + bottom ~ variable,
  value.var = '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
)

```

```

# convert long -> wide
dcast(
  as.data.table(a),
  formula = id + top + bottom ~ variable,
  value.var = '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

```
slice.fast(object, fm, top.down = TRUE, just.the.data = FALSE, strict = TRUE)
```

Arguments

object	a SoilProfileCollection
fm	A formula: either <code>integer.vector ~ var1 + var2 + var3</code> where named variables are sliced according to <code>integer.vector</code> OR where all variables are sliced according to <code>integer.vector</code> : <code>integer.vector ~ ..</code>
top.down	logical, slices are defined from the top-down: <code>0:10</code> implies 0-11 depth units.
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.

Details

By default, slices are defined from the top-down: 0:10 implies 0-11 depth units.

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

```
library(aqp)

# simulate some data, IDs are 1:20
d <- lapply(1:20, random_profile)
d <- do.call('rbind', d)

# 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 - 26 cm
# note that slices are specified by default as "top-down"
# e.g. the lower depth will always be top + 1
s <- slice(d, 0:25 ~ name + p1 + p2 + p3)
par(mar=c(0,1,0,1))
plot(s)
```

```

# generate slices from 0 - 11 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$sp1[1:10] <- NA
s <- slice(d, 10 ~ ., just.the.data=TRUE)
print(s)

## Not run:
##
## 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() !!!')

## End(Not run)

```

slicedHSD

Tukey's HSD Over Slices

Description

Apply Tukey's HSD over 1-unit depth slices.

Usage

```
slicedHSD(object, fm, conf = 0.95)
```


Arguments

object	SoilProfileCollection object
fm	a formula describing depth sequence, horizon attribute, and site (grouping) attribute. For example 0:100 ~ estimated_oc taxonname
conf	confidence applied in TukeyHSD

Author(s)

D.E. Beaudette and Sharon Perrone

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

See the [related tutorial](#).

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)
```

`soilPalette`*Soil Color Palette*

Description

A very simple function for generating labeled swatches of soil colors. Largely based on `colorspace::swatchplot`.

Usage

```
soilPalette(  
  colors,  
  lab,  
  lab.cex = 0.75,  
  dynamic.labels = TRUE,  
  x.inset = 0.01,  
  y.inset = 0.01,  
  ...  
)
```

Arguments

<code>colors</code>	vector of hex colors (e.g. <code>#A66E46FF</code>)
<code>lab</code>	vector of labels
<code>lab.cex</code>	character scaling for labels
<code>dynamic.labels</code>	logical, adjust label colors for maximum contrast via <code>invertLabelColor</code>
<code>x.inset</code>	horizontal adjustment for labels
<code>y.inset</code>	vertical adjustment for labels
<code>...</code>	further arguments to <code>colorspace::swatchplot</code>

Note

The result is a simple figure on the active plotting device.

Author(s)

D.E. Beaudette

Examples

```
# maybe useful for teaching about soil color  
  
par(mfrow=c(2,1), mar=c(1,1,1,1))  
  
# demonstrate range of Munsell value  
m <- sprintf('10YR %s/4', 2:8)
```

```

# convert to hex representation
cols <- parseMunsell(m)
# plot
soilPalette(cols, m)

# demonstrate range of Munsell chroma
m <- sprintf('10YR 4/%s', 2:8)
# convert to hex representation
cols <- parseMunsell(m)
# plot
soilPalette(cols, m)

```

SoilProfileCollection *An S4 object representation of a group of soil profiles.*

Description

In general, one should use `depths()` to initiate a `SoilProfileCollection` object from data. However, sometimes there are instances where either an empty, or very specific, object is needed. If that is the case, the general constructor `SoilProfileCollection` is available.

Usage

```

SoilProfileCollection(
  idcol = "id",
  hzidcol = "hzID",
  depthcols = c("top", "bottom"),
  metadata = list(aqp_df_class = "data.frame", aqp_group_by = "", aqp_hzdesgn = "",
    aqp_hztexcl = "", stringsAsFactors = FALSE),
  horizons = data.frame(id = character(0), hzID = character(0), top = numeric(0),
    bottom = numeric(0), stringsAsFactors = FALSE),
  site = data.frame(id = character(0), stringsAsFactors = FALSE),
  sp = new("SpatialPoints"),
  diagnostic = data.frame(stringsAsFactors = FALSE),
  restrictions = data.frame(stringsAsFactors = FALSE)
)

```

Arguments

<code>idcol</code>	character Profile ID Column Name
<code>hzidcol</code>	character Horizon ID Column Name
<code>depthcols</code>	character, length 2 Top and Bottom Depth Column Names
<code>metadata</code>	list, metadata including data.frame class in use and depth units
<code>horizons</code>	data.frame An object inheriting from data.frame containing Horizon data.
<code>site</code>	data.frame An object inheriting from data.frame containing Site data.

sp	SpatialPoints A SpatialPoints object. Generally initialized with coordinates(spc) <~ x + y.
diagnostic	data.frame An object inheriting from data.frame containing diagnostic feature data. Must contain profile ID. See diagnostic_hz()
restrictions	data.frame An object inheriting from data.frame containing restrictive feature data. Must contain profile ID. See restrictions()

Slots

idcol character.
 hzidcol character.
 depthcols character.
 metadata list.
 horizons data.frame.
 site data.frame.
 sp SpatialPoints.
 diagnostic data.frame.
 restrictions data.frame.

Author(s)

Pierre Roudier, Dylan E. Beaudette, Andrew G. Brown

Examples

```
## structure of default, empty SoilProfileCollection
str(SoilProfileCollection())

## use the depths() formula interface to specify
## profile ID, top and bottom depth and set up
## a SPC that is topologically correct and complete

d <- do.call('rbind',lapply(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 <- pbindlist(list(d.345, d.1, d.2))
plot(d.new)
```

```

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

## horizon designation column
hzdesgnname(sp1) <- "name"
hzdesgnname(sp1)

## all designations in an SPC (useful for single profile SPC)
hzDesgn(sp1)

## horizon texture class column
hztexclname(sp1) <- "texture"
hztexclname(sp1)

## 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 merge (left-join) of new columns and
# replacement of existing columns via horizons<-
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)
```

soiltexture

Lookup tables for sand, silt, clay, texture class, and textural modifiers.

Description

A list that contains a snapshot of the values generated using the logic from the particle size estimator calculation in NASIS, the average values per texture class, and average rock fragment values by textural modifier.

A list that contains a snapshot of the values generated using the logic from the particle size estimator calculation in NASIS, the average values per texture class, and average rock fragment values by textural modifier.

Format

A list with 3 data frames. The first named values which contains values for sand, silt and clay by texture class. The second with average values for sand, silt and clay per texture class. The third has fragvoltot low, rv and high values for texmod.

list("clay") clay percentage of the fine earth fraction, a integer vector

list("sand") sand percentage of the fine earth fraction, a integer vector

list("silt") silt percentage of the fine earth fraction, a integer vector

list("texcl") texture class, a character vector

list("texmod") textural modifiers, a character vector

A list with 3 data frames. The first named values which contains values for sand, silt and clay by texture class. The second with average values for sand, silt and clay per texture class. The third has fragvoltot low, rv and high values for texmod.

list("clay") clay percentage of the fine earth fraction, a integer vector

list("sand") sand percentage of the fine earth fraction, a integer vector

list("silt") silt percentage of the fine earth fraction, a integer vector

list("texcl") texture class, a character vector

list("texmod") textural modifiers, a character vector

Details

A list that contains a snapshot of the values generated using the logic from the particle size estimator calculation in NASIS, and the average values per texture class.

A list that contains a snapshot of the values generated using the logic from the particle size estimator calculation in NASIS, and the average values per texture class.

SoilTextureLevels	<i>Ranking Systems for USDS Soil Texture Classes</i>
-------------------	--

Description

Generate a vector of USDA soil texture codes or class names, sorted according to approximate particle size

Usage

```
SoilTextureLevels(which = "codes")
```

Arguments

which 'codes' (texture codes) or 'names' (texture class names)

Value

an ordered factor

References

[Field Book for Describing and Sampling Soils, version 3.0](#)

Examples

```
# class codes
SoilTextureLevels()

# class names
SoilTextureLevels(which = 'names')
```

`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

https://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.

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.

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.

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(data.table)

  a.wide.q25 <- dcast(a, top + bottom ~ variable, value.var = c('p.q25'))
  a.wide.q50 <- dcast(a, top + bottom ~ variable, value.var = c('p.q50'))
  a.wide.q75 <- dcast(a, top + bottom ~ variable, value.var = 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))

# 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

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

```
# load sample data set, a simple data.frame object with horizon-level data from 10 profiles
library(aqp)
data(sp4)
str(sp4)
sp4$idbak <- sp4$id
#sp4 <- sp4[order(match(sp4$id, aqp:::coalesce.idx(sort(sp4$id))), sp4$top),]

# 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

# check integrity of site:horizon linkage
spc_in_sync(sp4)

# check horizon depth logic
checkHzDepthLogic(sp4)

# 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
```

```

# add attributes to metadata list
metadata(sp4)$describer <- 'DGM'
metadata(sp4)$date <- as.Date('2009-01-01')
metadata(sp4)$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.'

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)

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 .. ..$ CaCO3 : CaCO3 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)

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')
```

spc2mpspline,SoilProfileCollection-method

*Missing-data-safe, SPC-wide wrapper around mpspline2::mpspline
"continuous" 1cm output*

Description

Facilitate safe use of just about any numeric SPC horizon attribute, from any SPC, with `mpspline2::mpspline`. Currently only works with a single attribute. This function will automatically filter profiles with NA in attribute of interest which may be more conservative filtering than you expect. The intention here is that a SPC of related profile instances could be splined, and then the spline results aggregated over the full interval where data was available.

Data completeness is assessed and the input SPC is filtered and truncated to create a container for the 1cm results from `mpspline2::mpspline`.

Usage

```
## S4 method for signature 'SoilProfileCollection'
spc2mpspline(
  object,
  var_name = NULL,
  pattern = "R|Cr|Cd|qm",
```



```

    hzdesgn = guessHzDesgnName(object),
    ...
  )

```

Arguments

object	A SoilProfileCollection
var_name	Column name in @horizons slot of object containing numeric values to spline
pattern	Regex pattern to match for bottom of profile (passed to estimateSoilDepth) default: "RlCr Cdlqm"
hzdesgn	Column name in @horizons slot of object containing horizon designations default: aqp::guessHzDesgnName(object)
...	Additional arguments to mpspline2::mpspline

Value

A SoilProfileCollection with 1cm slices. Spline variables are in columns prefixed with "spline_" and RMSE/RMSE_IQR are in columns prefixed with "rmse_". If any profiles were removed from the collection, their profile IDs are stored in attr(result, 'removed').

Author(s)

Andrew G. Brown

Examples

```

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

res <- spc2mpspline(sp1, "prop")

plotSPC(res[1:5,], color = "prop_spline", divide.hz = FALSE)

```

spc_in_sync

Quickly assess relative state of site and horizon slots

Description

Determine "state" of SoilProfileCollection before or after major modifications of site or horizon slot contents.

Two logical checks are performed on the site and horizon tables, and a third element valid returns TRUE when both checks are TRUE.

Check 1: Same number of sites in site as number of sites in horizons. No intermingling of IDs, no orphan horizons, no sites without horizons (for now)

Check 2: Site IDs match coalesced profile ID from horizons. Ensures the same *relative* ordering, but horizons still may be out of order within profiles

Usage

```
spc_in_sync(object)
```

Arguments

object A SoilProfileCollection

Value

data.frame

Author(s)

Andrew G. Brown

data(sp5)

spc_in_sync(sp5)

split,SoilProfileCollection-method

Split a SoilProfileCollection object into a list of SoilProfileCollection objects.

Description

This function splits a SoilProfileCollection into a list of SoilProfileCollection objects using a site-level attribute to define groups or profile ID (idname(x)).

Usage

```
## S4 method for signature 'SoilProfileCollection'
split(x, f, drop = TRUE, ...)
```

Arguments

x a SoilProfileCollection object

f a character vector naming a single site-level attribute that defines groups, a ‘factor’ in the sense that as .factor(f) defines the grouping, or a list of such factors in which case their interaction is used for the grouping.

drop logical indicating if levels that do not occur should be dropped (if f is a factor or a list).

... Additional arguments are ignored

Details

As of aqp 1.25, omission of `f` argument is no longer possible, as the base R generic is overloaded by this `SoilProfileCollection` method. This used to result in an "identity" split, according to `idname(x)`, e.g. a list as long as `length(x)`, with a single-profile `SoilProfileCollection` per list element. Replicate this behavior using `f = idname(x)` or `f = profile_id(x)`

Value

A list of `SoilProfileCollections` or `NULL` for empty result.

Author(s)

D.E Beaudette

Examples

```
data(sp2)
depths(sp2) <- id ~ top + bottom

# add a more interesting site-level attribute
site(sp2) <- ~ surface

# using identity site-level attribute (profile ID)
p1 <- split(sp2, f = idname(sp2))
names(p1)
length(p1)

# using vector equal in length to number of profiles (profile ID, again)
p2 <- split(sp2, f = profile_id(sp2))
names(p2)
length(p2)

# which are both equivalent to setting `f` to NULL
p3 <- split(sp2, f = NULL)
names(p3)
length(p3)

# split on surface (age) site-level var
p4 <- split(sp2, f = "surface")
names(p4)
length(p4) # 5 unique "surfaces", 5 SPCs in result list
```

Description

Uses checkHzDepthLogic to identify presence of depth logic errors, same depths, missing depths, and overlaps/gaps between the horizons of each profile in a SoilProfileCollection.

Usage

```
splitLogicErrors(object, interact = FALSE, ...)
```

Arguments

object	A SoilProfileCollection
interact	Calculate interaction between the four logic errors for groups? Default: FALSE always returns 4 groups, one for each logic error type.
...	Additional arguments to split.default, called when interact = TRUE

Value

A named list of SoilProfileCollections (or NULL), with names: "depthLogic", "sameDepth", "missingDepth", "overlapOrGap". If interact = TRUE then the list elements groups determined by interaction() of the error types.

Examples

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

# no errors (all four list elements return NULL)
splitLogicErrors(sp4)

# NA in top depth triggers depth logic and missing depth errors
data(sp4)
sp4$top[1] <- NA
depths(sp4) <- id ~ top + bottom

splitLogicErrors(sp4)

# interact = TRUE gets errors for profile 1 in same group
# and allows you to pass extra arguments to split.default()
splitLogicErrors(sp4, interact = TRUE, sep = "_", drop = TRUE)
```

subApply	<i>Subset SPC based on result of performing function on each profile</i>
----------	--

Description

subApply() is a function used for subsetting SoilProfileCollections. It currently does NOT support for "tidy" lexical features in the ... arguments passed to profileApply(). The expectation is that the function .fun takes a single-profile SoilProfileCollection and returns a logical value of length one. The use case would be for any logical comparisons that cannot be evaluated inline by subSPC() because they require more than simple logical operations.

Arguments

object	A SoilProfileCollection
.fun,	A function that takes a single profile, returns <i>logical</i> of length 1.
...	Additional arguments are passed to .fun

Value

A SoilProfileCollection.

Author(s)

Andrew G. Brown.

subset, SoilProfileCollection-method	<i>Subset a SoilProfileCollection with logical expressions</i>
--------------------------------------	--

Description

subset() is a function used for subsetting SoilProfileCollections. It allows the user to specify an arbitrary number of logical vectors (equal in length to site or horizon), separated by commas. The function includes some support for non-standard evaluation found in the tidyverse. This greatly simplifies access to site and horizon-level variable compared to subset.default, as '\$' or '[' methods are not needed.

Usage

```
## S4 method for signature 'SoilProfileCollection'
subset(x, ..., greedy = FALSE)
```

```
## S4 method for signature 'SoilProfileCollection'
filter(.data, ..., .preserve = FALSE)
```

Arguments

<code>x</code>	A <code>SoilProfileCollection</code>
<code>...</code>	Comma-separated set of R expressions that evaluate as TRUE or FALSE. Length for individual expressions matches number of sites OR number of horizons, in object.
<code>greedy</code>	Use "greedy" matching for combination of site and horizon level matches? <code>greedy = TRUE</code> is the union, whereas <code>greedy = FALSE</code> (default) is the intersection (of site and horizon matches).
<code>.data</code>	A <code>SoilProfileCollection</code> (<code>filter</code> method only, in lieu of <code>x</code>)
<code>.preserve</code>	Relevant when the <code>.data</code> input is grouped. Not (yet) implemented for <code>SoilProfileCollection</code> objects.

Details

In base R, the method that performs extraction based on a set of expressions is `subset`, so this is the "default" name in the AQP package. The `filter` method is defined in the base R `stats` package for linear filtering of time series. If you need to use the base method after loading `aqp`, you can use `stats::filter` to be explicit about which function you want. We mirror the `dplyr` package in over-loading the `filter` method with a generic that allows for arbitrary number of vectors (`...`) to be included in the "filter," which means you may also need to use `aqp::filter` or `dplyr::filter` as appropriate depending on the order you load packages or your conflict resolution option settings.

Value

A `SoilProfileCollection`.

Author(s)

Andrew G. Brown.

See Also

[filter](#)

`subsetProfiles` *DEPRECATED use subset*

Description

This function is used to subset `SoilProfileCollection` objects using either site-level or horizon-level attributes, or both.

Usage

```
## S4 method for signature 'SoilProfileCollection'
subsetProfiles(object, s, h, ...)
```

Arguments

object	object
s	fully-quoted search criteria for matching via site-level attributes
h	fully-quoted search criteria for matching via horizon-level attributes
...	not used

Details

The `s` argument supplies a fully-quoted search criteria for matching via site or horizon-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.

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, ])
```

summarizeSPC

Perform summaries on groups (from group_by) and create new site or horizon level attributes

Description

`summarize()` is a function used for summarizing `SoilProfileCollections`. Specify the groups using the `group_by` verb, and then (named) expressions to evaluate on each group. The result is a `data.frame` with one row per categorical level in the grouping variable and one column for each summary variable.

Usage

```
summarizeSPC(object, ...)
```

Arguments

object	A SoilProfileCollection
...	A set of (named) comma-delimited R expressions that resolve to a summary value. e.g groupmean = mean(clay, na.rm = TRUE)

Value

A data.frame with one row per level in the grouping variable, and one column for each summary

Author(s)

Andrew G. Brown

tauW	<i>Compute weighted naive and tau statistics for a cross-classification matrix</i>
------	--

Description

tauW: Computes: (1) unweighted naive, (2) weighted naive, (3) unweighted *tau*, (4) weighted *tau* accuracy statistics

Usage

```
tauW(
  CM,
  W = diag(sqrt(length(as.matrix(CM)))),
  P = rep(1/nrow(as.matrix(CM)), nrow(as.matrix(CM)))
)
```

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

Details

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.

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 $P = \emptyset$ is interpreted as $P = \text{column marginals}$.

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. doi: [10.1016/j.geoderma.2017.01.012](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. doi: [10.1016/09242716\(96\)000074](https://doi.org/10.1016/09242716(96)000074)

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)

```

texcl_to_ssc

Textural conversions

Description

These functions consist of several conversions between sand, silt and clay to texture class and visa versa, textural modifiers to rock fragments, and grain size composition to the family particle size class.

Usage

```
texcl_to_ssc(texcl = NULL, clay = NULL, sample = FALSE)
```

```
ssc_to_texcl(sand = NULL, clay = NULL, as.is = FALSE, droplevels = TRUE)
```

```
texmod_to_fragvoltot(texmod = NULL, lieutex = NULL)
```

```
texture_to_taxpartsize(
```

```

    texcl = NULL,
    clay = NULL,
    sand = NULL,
    fragvoltot = NULL
  )

```

Arguments

texcl	vector of texture classes than conform to the USDA code conventions (e.g. c C, s l SIL, s l SL, c o s COS)
clay	vector of clay percentages#'
sample	logical: should ssc be random sampled from the lookup table? (default: FALSE)
sand	vector of sand percentages
as.is	logical: should character vectors be converted to factors? (default: TRUE)
droplevels	logical: indicating whether to drop unused levels in factors. This is useful when the results have a large number of unused classes, which can waste space in tables and figures.
texmod	vector of textural modifiers that conform to the USDA code conventions (e.g. gr GR, grv GRV)
lieutex	vector of in lieu of texture terms that conform to the USDA code conventions (e.g. gr GR, pg PG), only used when fragments or artifacts are > 90 percent by volume (default: NULL)
fragvoltot	vector of rock fragment percentages

Details

These functions are intended to estimate missing values or check existing values. The `ssc_to_texcl()` function uses the same logic as the particle size estimator calculation in NASIS to classify sand and clay into texture class. The results are stored in `soiltexture` and used by `texcl_to_ssc()` as a lookup table to convert texture class to sand, silt and clay. The function `texcl_to_ssc()` replicates the functionality described by Levi (2017).

When `sample = TRUE`, the results can be used to estimate within-class, marginal distributions of sand, silt, and clay fractions. It is recommended that at least 10 samples be drawn for reasonable estimates.

Unlike the other functions, `texture_to_taxpartsize()` is intended to be computed on weighted averages within the family particle size control section. Also recall from the criteria that carbonate clay should be subtracted from clay content and added to silt content. Similarly, if the percent of very fine sand is known it should be subtracted from the sand, and added to the silt content. Unlike the other functions, `texture_to_taxpartsize()` is intended to be computed on weighted averages within the family particle size control section. Also recall from the criteria that carbonate clay should be subtracted from clay content and added to silt content. Similarly, if the percent of very fine sand is known it should be subtracted from the sand, and added to the silt content.

Value

- `texcl_to_ssc`: A `data.frame` containing columns "sand", "silt", "clay"

- ssc_to_texcl: A character vector containing texture class
- texmod_to_fragvoltot: A numeric vector containing total fragment volume
- texture_to_taxpartsize: a character vector containing "taxpartsize" classes

Author(s)

Stephen Roecker

References

Matthew R. Levi, Modified Centroid for Estimating Sand, Silt, and Clay from Soil Texture Class, Soil Science Society of America Journal, 2017, 81(3):578-588, ISSN 1435-0661, doi: [10.2136/sssaj2016.09.0301](https://doi.org/10.2136/sssaj2016.09.0301).

Examples

```
# example of ssc_to_texcl()
tex <- expand.grid(sand = 0:100, clay = 0:100)
tex <- subset(tex, (sand + clay) < 101)
tex$ssc <- ssc_to_texcl(sand = tex$sand, clay = tex$clay)
head(tex)

# example of texcl_to_ssc(texcl)
texcl <- c("cos", "s", "fs", "vfs", "lcos", "ls",
          "lfs", "lvfs", "cosl", "sl", "fsl", "vfsl", "l",
          "sil", "si", "scl", "cl", "sicl", "sc", "sic", "c"
          )
test <- texcl_to_ssc(texcl)
head(test <- cbind(texcl, test), 10)

# example of texcl_to_ssc(texcl, clay)
data(soiltexture)
st <- soiltexture$values
idx <- sample(1:length(st$ssc), 10)
st <- st[idx, ]
ssc <- texcl_to_ssc(texcl = st$ssc)
head(cbind(texcl = st$ssc, clay = ssc$clay))

# example of texmod_to_fragvoltot()
frags <- c("gr", "grv", "grx", "pgr", "pgrv", "pgrx")
head(texmod_to_fragvoltot(frag))

# example of texture_to_taxpartsize()
tex <- data.frame(texcl = c("c", "cl", "l", "ls", "s"),
                 clay = c(55, 33, 18, 6, 3),
                 sand = c(20, 33, 42, 82, 93),
```

```

      fragvoltot = c(35, 15, 34, 60, 91))
tex$fpvc <- texture_to_taxpartsize(texcl = tex$texcl,
                                clay = tex$clay,
                                sand = tex$sand,
                                fragvoltot = tex$fragvoltot)

head(tex)

# example of texture_to_taxpartsize() with carbonate clay and very fine sand
carbclay <- rnorm(5, 2, 3)
vfs <- rnorm(5, 10, 3)
st$fpvc <- texture_to_taxpartsize(texcl = tex$texcl,
                                clay = tex$clay - carbclay,
                                sand = tex$sand - vfs,
                                fragvoltot = tex$fragvoltot)

head(tex)

# example of sample = TRUE
texcl <- rep(c("cl", "sil", "sl"), 10)
ssc1 <- cbind(texcl, texcl_to_ssc(texcl = texcl, sample = FALSE))
ssc2 <- cbind(texcl, texcl_to_ssc(texcl = texcl, sample = TRUE))
ssc1$sample <- FALSE
ssc2$sample <- TRUE
ssc <- rbind(ssc1, ssc2)
aggregate(clay ~ sample + texcl, data = ssc, summary)

```

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. 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,
  rv.col = "red",
  range.border = "black",

```

```

range.col = "RoyalBlue",
range.alpha = 80,
range.lty = 1,
range.lwd = 2,
main = "Soil Textures",
legend.cex = 0.75,
legend = TRUE,
...
)

```

Arguments

ssc	data.frame with columns: 'SAND', 'SILT', 'CLAY', values are percentages that should add to 100. No NA allowed.
p	vector of percentiles (length = 3) defining 'low', 'representative value', and 'high'
delta	grid size used to form low-high region
rv.col	color used for representative value (RV) symbol
range.border	color used for polygon border enclosing the low-high region
range.col	color used for polygon enclosing the low-high region
range.alpha	transparency of the low-high range polygon (0-255)
range.lty	line style for polygon enclosing the low-high region
range.lwd	line weight polygon enclosing the low-high region
main	plot title
legend.cex	scaling factor for legend
legend	logical, enable/disable automatic legend
...	further arguments passed to soiltexture::TT.points

Value

an invisible matrix with marginal percentiles of sand, silt, and clay

Author(s)

D.E. Beaudette, J. Nemecek, K. Godsey

See Also

[bootstrapSoilTexture](#)

Examples

```
if(
```

```

requireNamespace("Hmisc") &
  requireNamespace("compositions") &
  requireNamespace("soiltexture")
) {

  # sample data
  data('sp4')

  # subset rows / columns
  ssc <- sp4[grep('^Bt', sp4$name), c('sand', 'silt', 'clay')]
  names(ssc) <- toupper(names(ssc))

  # make figure, marginal percentiles are silently returned
  stats <- textureTriangleSummary(
    ssc, pch = 1, cex = 0.5,
    range.alpha = 50,
    range.lwd = 1,
    col = grey(0.5),
    legend = FALSE
  )

  # check
  stats

  # simulate some data and try again
  s <- bootstrapSoilTexture(ssc, n = 100)$samples

  # make the figure, ignore results
  textureTriangleSummary(
    s, pch = 1, cex = 0.5,
    range.alpha = 50,
    range.lwd = 1,
    col = grey(0.5),
    legend = FALSE
  )
}

```

 thompson.bell.darkness

Thompson-Bell (1996) Index

Description

Calculate the "Profile Darkness Index" by the method of Thompson & Bell (1996) "Color index for identifying hydric conditions for seasonally saturated mollisols in Minnesota" DOI: 10.2136/sssaj1996.03615995006000060051x. The Thompson-Bell Index has been shown to reflect catenary relationships in some Mollisols of Minnesota (generally: wetter landscape positions = thicker, darker surfaces).

Usage

```
thompson.bell.darkness(
  p,
  name = NULL,
  pattern = "^A",
  value = "m_value",
  chroma = "m_chroma"
)
```

Arguments

p	A single-profile SoilProfileCollection (e.g. via profileApply())
name	Column name containing horizon designations used to find A horizons (default: first column name containing 'name')
pattern	Regular expression to match A horizons (default: "^A" which means horizon designation <i>starts with A</i>)
value	Column name containing horizon color values (default: "m_value")
chroma	Column name containing horizon color chromas (default: "m_chroma")

Value

A numeric vector reflecting horizon darkness (lower values = darker).

Author(s)

Andrew G. Brown.

References

Thompson, J.A. and Bell, J.C. (1996), Color Index for Identifying Hydric Conditions for Seasonally Saturated Mollisols in Minnesota. *Soil Science Society of America Journal*, 60: 1979-1988. doi:10.2136/sssaj1996.03615995006000060051x

transform,SoilProfileCollection-method

Transform a SPC with expressions based on site or horizon level attributes

Description

transform() is a function used for modifying columns in SoilProfileCollections.

It allows the user to specify an arbitrary number of expressions that resolve to the (re-)calculation of one or more site or horizon level attributes. For instance: mutate(spc, thickness = hzdepb - hzdept). The expressions may depend on one another, and are evaluated from left to right.

Usage

```
## S4 method for signature 'SoilProfileCollection'
transform(`_data`, ...)
```

Arguments

<code>_data</code>	A SoilProfileCollection
<code>...</code>	Comma-separated set of R expressions e.g. <code>thickness = hzdepb - hzdept, hzdepn = hzdept + round(thk / 2)</code>

Value

A SoilProfileCollection

Author(s)

Andrew G. Brown.

```
trunc, SoilProfileCollection-method
```

Truncate a SoilProfileCollection to specified top and bottom depth

Description

`trunc` is a wrapper method around `glomApply` for the case when the same top and bottom depth is required for all profiles in a collection. In contrast, `glomApply` allows for arbitrary functions to be run on each profile to calculate a unique set of depths.

Usage

```
## S4 method for signature 'SoilProfileCollection'
trunc(x, z1, z2)
```

Arguments

<code>x</code>	A SoilProfileCollection
<code>z1</code>	Upper boundary
<code>z2</code>	Lower boundary

Value

A SoilProfileCollection truncated to interval `[z1, z2]`

Examples

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

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

### TRUNCATE all profiles in sp3 to [0,25]

# set up plot parameters
par(mfrow=c(2,1), mar=c(0,0,0,0))

# full profiles
plot(sp3)

# trunc'd profiles
plot(trunc(sp3, 0, 25))
```

 unique

Get the indexes of unique profiles in a SoilProfileCollection

Description

Calculate MD5 hash of each profile in a SoilProfileCollection for the specified variables.

Usage

```
## S4 method for signature 'SoilProfileCollection'
unique(x, vars)
```

Arguments

x	a SoilProfileCollection
vars	Variables to consider in uniqueness.

Examples

```
data(sp5)

# find indices where all specified vars are unique
# these match the indices of all profiles in sp5
# therefore, all profiles in sp5 are unique

all(unique(sp5, vars=c("id", "sand", "silt", "clay")) == 1:length(sp5))
```

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)

```

us.state.soils	<i>US State Soils</i>
----------------	-----------------------

Description

A listing of the 50 US state soils, along with Puerto Rico and Virgin Islands.

Usage

```
data(us.state.soils)
```

Format

state state name
abbreviated abbreviated state name
series soil series name

validSpatialData	<i>Get names of columns in site table</i>
------------------	---

Description

Are the contents of @sp valid: n x 2 matrix? If not, then contents of @sp in the SoilProfileCollection are an empty SpatialPoints object.

Arguments

object a SoilProfileCollection

xtableTauW	<i>Format a LaTeX table with results</i>
------------	--

Description

Format a LaTeX table with results

Usage

```
xtableTauW(result.tau, file.name = "tau_results_table.tex")
```

Arguments

result.tau	results returned by tauW
file.name	name of file to write output TeX file; Default: file.name="tau_results_table.tex"

[,SoilProfileCollection-method

Matrix/data.frame-like access to profiles and horizons in a SoilProfileCollection

Description

You can access the contents of a SoilProfileCollection by profile and horizon "index", *i* and *j*, respectively: `spc[i, j, ...]`. Subset operations are propagated to other slots (such as diagnostics or spatial) when they result in removal of sites from a collection.

- *i* refers to the profile position within the collection. By default the order is based on the C SORT order of the variable that you specified as your unique profile ID at time of object construction. Note that if your ID variable was numeric, then it has been sorted as a character.
- *j* refers to the horizon or "slice" index. This index is most useful when either a) working with `slice'd` SoilProfileCollection or b) working with single-profile collections. *j* returns the layer in the specified index positions for all profiles in a collection.
- ... is an area to specify an expression that is evaluated in the subset. Currently supported
 - `.LAST` (last horizon in each profile): return the last horizon from each profile. This uses *i* but ignores the regular *j* index.
 - `.FIRST` (first horizon in each profile): return the last horizon from each profile. This uses *i* but ignores the regular *j* index.
 - `.HZID` (horizon index not SoilProfileCollection result): return the horizon indices corresponding to *i+j+...* ("k") constraints

Usage

```
## S4 method for signature 'SoilProfileCollection'
x[i, j, ..., drop = TRUE]
```

Arguments

x	a SoilProfileCollection
i	a numeric or logical value denoting profile indices to select in a subset
j	a numeric or logical value denoting horizon indices to select in a subset
...	non-standard expressions to evaluate in a subset
drop	not used

[[*Get column of horizon or site data in a SoilProfileCollection*

Description

Get the data from a column accessed by name. Column names other than profile ID are not shared between site and horizons. Bonus: [[gives access to all site and horizon level variables in tab complete for RStudio using the magrittr pipe operator!

Usage

```
## S4 method for signature 'SoilProfileCollection,ANY,ANY'
x[[i, j]]
```

Arguments

x	a SoilProfileCollection
i	an expression resolving to a single column name in site or horizon table
j	(not used)

Examples

```
data(sp2)
depths(sp2) <- id ~ top + bottom
site(sp2) <- ~ surface

# get with [[
sp2[['surface']]

# get using "unknown" expression:
# "2nd + 3rd horizon column names"
for(i in horizonNames(sp2)[2:3])
  print(sp2[[i]])

data(sp5)

# some column names to work with
rgb.columns <- c("R25", "G25", "B25")
```

```

res <- lapply(rgb.columns, function(x) {
  # [[ allows you to access column names in a loop
  round(sp5[[x]] * 255)
})

# rename scaled results
names(res) <- paste0(rgb.columns, "_scl")

# add horizon ID to results
result <- data.frame(hzID = hzID(sp5), do.call('cbind', res))
head(result)

# join result back into horizons
horizons(sp5) <- result

```

[[<- *Add or change column of horizon or site data in a SoilProfileCollection*

Description

Add or change the data from a column accessed by name. Column names other than profile ID are not shared between site and horizons. The benefit of using double bracket setter over \$ is that name can be calculated, whereas with \$, it must be known a priori and hard coded.

When using the double bracket setter the length of input and output matching either the number of sites or number of horizons is used to determine which slot new columns are assigned to.

Usage

```

## S4 replacement method for signature 'SoilProfileCollection,ANY,ANY'
x[[i]] <- value

```

Arguments

x	a SoilProfileCollection
i	an expression resolving to a single column name in site or horizon table-
value	New value to replace – unit length or equal in length to number of sites or horizons in the collection.

\$ *Get data from column of horizon or site data in a SoilProfileCollection*

Description

Get the data from a column accessed by name `x$name`. Column names other than profile ID are not shared between site and horizons.

Usage

```
## S4 method for signature 'SoilProfileCollection'
x$name
```

Arguments

<code>x</code>	a <code>SoilProfileCollection</code>
<code>name</code>	a single column name in site or horizon table

Examples

```
data(sp1)

depths(sp1) <- id ~ top + bottom

# get data from a column by name (prop)
sp1$prop
```

\$<- *Set data in column of horizon or site data in a SoilProfileCollection*

Description

Set the data in a column accessed by name `spc$name`. Column names other than profile ID are not shared between site and horizons.

When using `$<-`, the length of input and output matching either the number of sites or number of horizons is used to determine which slot new columns are assigned to. Use `site(x)$name <-value` or `horizons(x)$name <-value` to be explicit about which slot is being accessed.

Usage

```
## S4 replacement method for signature 'SoilProfileCollection'
x$name <- value
```


Arguments

x	a SoilProfileCollection
name	a single column name in site or horizon table
value	Replacement values: unit length or equal to number of horizons or sites.

Index

- * **array**
 - tauW, 248
- * **datasets**
 - ca630, 32
 - equivalent_munsell, 59
 - jacobs2000, 126
 - munsell, 136
 - munsell.spectra, 138
 - pms.munsell.lut, 172
 - ROSETTA.centroids, 191
 - rowley2019, 193
 - rruff.sample, 196
 - sierraTransect, 201
 - soil_minerals, 225
 - soiltexture, 223
 - sp1, 227
 - sp2, 228
 - sp3, 230
 - sp4, 233
 - sp5, 236
 - sp6, 239
 - us.state.soils, 260
- * **hplots**
 - colorContrastPlot, 39
 - contrastChart, 44
 - groupedProfilePlot, 94
 - missingDataGrid, 132
 - plotMultipleSPC, 160
 - plotSPC, 164
 - soilPalette, 219
 - textureTriangleSummary, 253
- * **hplot**
 - panel.depth_function, 145
 - plot_distance_graph, 171
- * **manip**
 - argillic.clay.increase.depth, 24
 - colorContrast, 38
 - contrastChart, 44
 - contrastClass, 45
 - crit.clay.argillic, 48
 - duplicate, 56
 - estimatePSCS, 61
 - estimateSoilDepth, 63
 - evalGenHZ, 65
 - evalMissingData, 67
 - explainPlotSPC, 68
 - f.noise, 69
 - generalize.hz, 77
 - get.increase.depths, 79
 - get.increase.matrix, 80
 - get.ml.hz, 81
 - getSoilDepthClass, 87
 - guessGenHzLevels, 97
 - huePosition, 112
 - hzDistinctnessCodeToOffset, 117
 - hzTopographyCodeToLineType, 121
 - hzTopographyCodeToOffset, 122
 - hzTransitionProbabilities, 123
 - pc, 151
 - random_profile, 182
 - resample.twotheta, 187
 - slab-methods, 205
 - slice-methods, 214
 - texcl_to_ssc, 250
 - unroll, 259
- * **methods**
 - pc, 151
 - slab-methods, 205
 - slice-methods, 214
- * **package**
 - aqp-package, 6
 - .HSD, 8
 - .as.data.frame.aqp, 7
 - .data_dots, 7
 - .lpp (random_profile), 182
 - .makeEquivalentMunsellLUT, 8
 - .parseGrouped_formula, 10
 - [,SoilProfileCollection-method, 261

- [[, 262
- [[, SoilProfileCollection, ANY, ANY-method
([[), 262
- [[, SoilProfileCollection, ANY-method,
([[), 262
- [[<-, 263
- [[<-, SoilProfileCollection, ANY, ANY-method
([[<-), 263
- \$, 264
- \$, SoilProfileCollection-method (\$), 264
- \$<-, 264
- \$<-, SoilProfileCollection-method (\$<-),
264

- addBracket, 10, 13, 168
- addDiagnosticBracket, 11, 12
- addVolumeFraction, 13
- aggregateColor, 14
- aggregateSoilDepth, 17
- alignTransect, 18
- allocate, 19
- aqp (aqp-package), 6
- aqp-package, 6
- aqp.env (aqp-package), 6
- aqp_df_class, 23
- aqp_df_class, SoilProfileCollection-method
(aqp_df_class), 23
- aqp_df_class<- (aqp_df_class), 23
- aqp_df_class<-, SoilProfileCollection-method
(aqp_df_class), 23
- argillic.clay.increase.depth, 24
- as, 25
- as, SoilProfileCollection-method (as), 25

- barron.torrent.redness.LAB, 26, 110
- bootstrapSoilTexture, 26, 254
- brierScore, 29
- buntley.westin.index, 30, 110

- c (c, SoilProfileCollection-method), 31
- c, SoilProfileCollection-method, 31
- ca630, 7, 32
- checkHzDepthLogic, 35, 37, 114
- checkSPC, 37, 185
- colorContrast, 38, 40, 46, 57, 58, 113
- colorContrastPlot, 39, 39, 57, 58
- colorQuantiles, 41
- colorRamp, 167

- combine
(c, SoilProfileCollection-method),
31
- combine, list-method
(c, SoilProfileCollection-method),
31
- combine, SoilProfileCollection-method
(c, SoilProfileCollection-method),
31
- compositeSPC, 42
- compositeSPC, SoilProfileCollection-method
(compositeSPC), 42
- confusionIndex, 43
- contrastChart, 44
- contrastClass, 45
- coordinates, 46
- coordinates, SoilProfileCollection-method
(coordinates), 46
- coordinates<-, SoilProfileCollection-method
(coordinates), 46
- correctAWC, 47
- create_progress_bar, 152
- crit.clay.argillic, 48

- daisy, 66, 153
- denormalize, 49
- depth_units, 53
- depth_units, SoilProfileCollection-method
(depth_units), 53
- depth_units<- (depth_units), 53
- depth_units<-, SoilProfileCollection-method
(depth_units), 53
- depthOf, 50
- depths<-, SoilProfileCollection-method,
52
- depths<-
(depths<-, SoilProfileCollection-method),
52
- depths<-, data.frame-method
(depths<-, SoilProfileCollection-method),
52
- depthWeights, 53
- depthWeights, SoilProfileCollection-method
(depthWeights), 53
- diagnostic_hz, 54
- diagnostic_hz, SoilProfileCollection-method
(diagnostic_hz), 54
- diagnostic_hz<-, 54

- diagnostic_hz<- ,SoilProfileCollection-method glom
 (diagnostic_hz<-), 54
 dice, 55
 duplicate, 56

 equivalent_munsell, 57, 58, 59
 equivalentMunsellChips, 9, 57, 60
 estimateAWC, 60
 estimatePSCS, 61
 estimateSoilDepth, 63, 87, 88
 estimateSoilDepth(), 18
 evalGenHZ, 65
 evalMissingData, 67
 explainPlotSPC, 68, 168

 f.noise, 69
 fillHzGaps, 72, 114
 filter, 246
 filter
 (subset,SoilProfileCollection-method), 245
 filter,SoilProfileCollection-method
 (subset,SoilProfileCollection-method), 245
 findOverlap, 74
 fixOverlap, 75, 159, 168

 generalize_hz, 15, 77, 98, 124
 genhzTableToAdjMat
 (hzTransitionProbabilities), 123
 genSlabLabels (slab-methods), 205
 get.increase.depths, 79
 get.increase.matrix, 48, 80
 get.ml_hz, 66, 81
 get.slice (slice-methods), 214
 getArgillicBounds, 48, 83
 getCambicBounds, 85
 getClosestMunsellChip, 86
 getLastHorizonID, 87, 186
 getMineralSoilSurfaceDepth
 (getSurfaceHorizonDepth), 88
 getPlowLayerDepth
 (getSurfaceHorizonDepth), 88
 getSoilDepthClass, 64, 87
 getSurfaceHorizon
 (getSurfaceHorizonDepth), 88
 getSurfaceHorizonDepth, 88
 glom, 93
 (glom,SoilProfileCollection-method), 90
 glom,SoilProfileCollection-method, 90
 glomApply, 91, 92, 93
 glomApply,SoilProfileCollection-method
 (glomApply), 92
 grepSPC, 93
 grepSPC,SoilProfileCollection-method
 (grepSPC), 93
 group_by (groupSPC), 97
 groupedProfilePlot, 94
 groupSPC, 97
 guessGenHzLevels, 97
 guessHzAttrName, 99, 100, 101
 guessHzDesgnName, 99, 100, 101
 guessHzTexClName, 99, 100, 101

 harden.melanization, 102
 harden.rubification, 104
 harmonize
 (harmonize,SoilProfileCollection-method), 106
 harmonize,SoilProfileCollection-method, 106
 hasDarkColors, 108
 horizonColorIndices, 110
 horizonDepths (horizonDepths<-), 111
 horizonDepths,SoilProfileCollection-method
 (horizonDepths<-), 111
 horizonDepths<- , 111
 horizonDepths<- ,SoilProfileCollection-method
 (horizonDepths<-), 111
 horizonNames (horizonNames<-), 111
 horizonNames,SoilProfileCollection-method
 (horizonNames<-), 111
 horizonNames<- , 111
 horizonNames<- ,SoilProfileCollection-method
 (horizonNames<-), 111
 horizons, 112
 horizons,SoilProfileCollection-method
 (horizons), 112
 horizons<- (horizons), 112
 horizons<- ,SoilProfileCollection-method
 (horizons), 112
 huePosition, 39, 112
 hurst.redness, 110, 114
 HzDepthLogicSubset, 114
 hzDepthTests, 115

- hzDesign, 116
- hzDesign, SoilProfileCollection-method (hzDesign), 116
- hzdesignname, 116
- hzdesignname, SoilProfileCollection-method (hzdesignname), 116
- hzdesignname<-, 117
- hzdesignname<-, SoilProfileCollection-method (hzdesignname<-), 117
- hzDistinctnessCodeToOffset, 117, 166, 168, 183
- hzDistinctnessCodeToOffset(), 156, 157
- hzID, 119
- hzID, SoilProfileCollection-method (hzID), 119
- hzID<-, 119
- hzID<-, SoilProfileCollection-method (hzID<-), 119
- hzidname (hzidname<-), 119
- hzidname, SoilProfileCollection-method (hzidname<-), 119
- hzidname<-, 119
- hzidname<-, SoilProfileCollection-method (hzidname<-), 119
- hztexclname, 120
- hztexclname, SoilProfileCollection-method (hztexclname), 120
- hztexclname<-, 120
- hztexclname<-, SoilProfileCollection-method (hztexclname<-), 120
- hzTopographyCodeToLineType, 121
- hzTopographyCodeToOffset, 122, 122, 166
- hzTransitionProbabilities, 123

- idname, 125
- idname, SoilProfileCollection-method (idname), 125
- invertLabelColor, 125
- isoMDS, 66

- jacobs2000, 126

- L1_profiles, 127
- length, 128
- length, SoilProfileCollection-method (length), 128
- lunique, 129

- max, 130
- max, SoilProfileCollection-method (max), 130
- maxDepthOf (depthOf), 50
- metadata, 130
- metadata, SoilProfileCollection-method (metadata), 130
- metadata<- (metadata), 130
- metadata<-, SoilProfileCollection-method (metadata), 130
- min, 131
- min, SoilProfileCollection-method (min), 131
- minDepthOf (depthOf), 50
- missingDataGrid, 132
- mixMunsell, 133, 159
- mollic.thickness.requirement, 135
- mostLikelyHzSequence (hzTransitionProbabilities), 123
- munsell, 136
- munsell.spectra, 135, 138
- munsell2rgb, 139, 142, 167, 218
- munsell2spc (munsell2spc, SoilProfileCollection-method), 141
- munsell2spc, SoilProfileCollection-method, 141
- mutate (transform, SoilProfileCollection-method), 256
- mutate, SoilProfileCollection-method (transform, SoilProfileCollection-method), 256
- mutate_profile, 143
- mutate_profile, SoilProfileCollection-method (mutate_profile), 143

- names, 143
- names, SoilProfileCollection-method (names), 143
- nrow, 144
- nrow, SoilProfileCollection-method (nrow), 144

- overlapMetrics, 144

- panel.depth_function, 145
- parseMunsell, 142, 149
- pbindlist, 150

- pc, 151
- permute_profile (perturb), 155
- perturb, 155, 204
- perturb(), 203
- plot (plotSPC), 164
- plot, SoilProfileCollection, ANY-method (plotSPC), 164
- plot, SoilProfileCollection, ANY-method, plot.SoilProfileCollection (plotSPC), 164
- plot_distance_graph, 171
- plotColorMixture, 159
- plotColorQuantiles, 160
- plotMultipleSPC, 160, 180
- plotSPC, 11, 13, 14, 68, 95, 118, 122, 123, 164
- pms.munsell.lut, 172
- prepanel.depth_function (panel.depth_function), 145
- pretty, 167, 168
- previewColors, 174
- profile_compare, 172, 183
- profile_compare (pc), 151
- profile_compare, data.frame-method (pc), 151
- profile_compare, SoilProfileCollection-method (pc), 151
- profile_id (profile_id<-), 181
- profile_id, SoilProfileCollection-method (profile_id<-), 181
- profile_id<-, 181
- profile_id<-, SoilProfileCollection-method (profile_id<-), 181
- profileApply, 63, 64, 176
- profileApply, SoilProfileCollection-method (profileApply), 176
- profileGroupLabels, 162, 168, 179
- proj4string, SoilProfileCollection-method, 181
- proj4string<-, SoilProfileCollection, ANY-method, 181
- quantile, 208
- random_profile, 182, 204
- random_profile(), 157
- rebuildSPC, 37, 185
- reorderHorizons, 186
- reorderHorizons, SoilProfileCollection-method (reorderHorizons), 186
- repairMissingHzDepths, 186
- replaceHorizons<-, 187
- replaceHorizons<-, SoilProfileCollection-method (replaceHorizons<-), 187
- resample.twotheta, 70, 187
- restrictions, 189
- restrictions, SoilProfileCollection-method (restrictions), 189
- restrictions<-, SoilProfileCollection-method (restrictions<-), 189
- rgb, 139
- rgb2munsell, 142, 190
- ROSETTA.centroids, 191
- rowley2019, 193
- rruff.sample, 188, 196
- segment, 197
- shannonEntropy, 199
- show, 200
- show, SoilProfileCollection-method (show), 200
- sierraTransect, 201
- silhouette, 66
- sim, 203
- site, 204
- site, SoilProfileCollection-method (site), 204
- site<- (site), 204
- site<-, SoilProfileCollection-method (site), 204
- siteNames (siteNames<-), 205
- siteNames, SoilProfileCollection-method (siteNames<-), 205
- siteNames<-, 205
- siteNames<-, SoilProfileCollection-method (siteNames<-), 205
- slab, 81, 145, 146, 215
- slab (slab-methods), 205
- slab(), 18, 82
- slab, SoilProfileCollection-method (slab-methods), 205
- slab-methods, 205
- slab2 (slab-methods), 205
- slice, 133, 146, 153, 208
- slice (slice-methods), 214
- slice, SoilProfileCollection-method (slice-methods), 214
- slice-methods, 214
- slice.fast (slice-methods), 214

- slicedHSD, 216
- soil_minerals, 225
- soilColorSignature, 217
- soilPalette, 219
- SoilProfileCollection, 151, 152, 207, 220
- SoilProfileCollection-class
 - (SoilProfileCollection), 220
- soiltexture, 223
- SoilTextureLevels, 224
- sp1, 7, 146, 227
- sp2, 7, 172, 228
- sp3, 7, 230
- sp4, 7, 233
- sp5, 7, 236
- sp6, 239
- spc2mpspline
 - (spc2mpspline, SoilProfileCollection-method), 240
- spc2mpspline, SoilProfileCollection-method, 240
- spc_in_sync, 241
- spc_in_sync, SoilProfileCollection-method
 - (spc_in_sync), 241
- split, SoilProfileCollection-method, 242
- splitLogicErrors, 243
- ssc_to_texcl (texcl_to_ssc), 250
- subApply, 245
- subApply, SoilProfileCollection-method
 - (subApply), 245
- subset
 - (subset, SoilProfileCollection-method), 245
- subset, SoilProfileCollection-method, 245
- subsetProfiles, 246
- subsetProfiles, SoilProfileCollection-method
 - (subsetProfiles), 246
- summarize (summarizeSPC), 247
- summarizeSPC, 247
- summarizeSPC, SoilProfileCollection-method,
 - (summarizeSPC), 247
- summaryTauW (tauW), 248

- tauW, 248
- texcl_to_ssc, 250
- texmod_to_fragvoltot (texcl_to_ssc), 250
- texture_to_taxpartsize (texcl_to_ssc), 250
- textureTriangleSummary, 253
- thompson.bell.darkness, 255
- transform, SoilProfileCollection-method, 256
- trunc, 91, 93
- trunc
 - (trunc, SoilProfileCollection-method), 257
- trunc, SoilProfileCollection-method, 257
- unique, 258
- unique, SoilProfileCollection-method
 - (unique), 258
- unroll, 259
- us.state.soils, 260
- validSpatialData, 260
- validSpatialData, SoilProfileCollection-method
 - (validSpatialData), 260
- xtableTauW, 261