

Package ‘aqp’

March 27, 2012

Version 1.0

Date 2012-03-26

Title Algorithms for Quantitative Pedology

Author Dylan Beaudette <debeaudette@ucdavis.edu>, Pierre Roudier
<roudierp@landcareresearch.co.nz>

Maintainer Dylan Beaudette <debeaudette@ucdavis.edu>, Pierre Roudier
<roudierp@landcareresearch.co.nz>

Depends R (>= 2.10), methods, plyr, reshape, lattice, grid, cluster,sp, Hmisc, stringr

Suggests scales, RColorBrewer, colorspace, maptools, foreign, MASS,ape, plotrix, rgeos, soilDB, latticeExtra, digest

Description A collection of algorithms related to modeling of soil resources, soil classification, soil profile aggregation, and visualization.

License GPL (>= 2)

Repository CRAN

Repository/R-Forge/Project aqp

Repository/R-Forge/Revision 531

Date/Publication 2012-03-27 15:58:26

R topics documented:

aqp-package	2
ca630	3
f.noise	5
generalize.hz	9
munsell	10
munsell2rgb	11
panel.depth_function	13

plot_distance_graph	15
profileApply-methods	16
profile_compare-methods	19
random_profile	22
resample.twotheta	25
rruff.sample	26
slice-methods	27
soil.slot	28
SoilProfileCollection-class	35
SoilProfileCollection-plotting-methods	37
sp1	39
sp2	41
sp3	43
sp4	45
sp5	47
spatial_subset-methods	50
SPC-slab-methods	51
SPC-utils	53
test_hz_logic	54
unroll	56
Index	58

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.

Demo: `demo(aqp)`

Extended Examples: <http://casoilresource.lawr.ucdavis.edu/drupal/taxonomy/term/56>

Author(s)

Dylan E. Beaudette <debeaudette@ucdavis.edu>

References

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

See Also

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

 ca630

Soil Data from the Central Sierra Nevada Region of California

Description

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

Usage

`data(ca630)`

Format

List containing:

\$site : A data frame containing site information.

user_site_id national user site id

mlra the MLRA

county the county

ssa soil survey area

lon longitude, WGS84

lat latitude, WGS84

pedon_key national soil profile id

user_pedon_id local soil profile id

cntrl_depth_to_top control section top depth (cm)

cntrl_depth_to_bot control section bottom depth (cm)

sampled_taxon_name soil series name

\$lab : A data frame containing horizon information.

pedon_key national soil profile id

layer_key national horizon id

layer_sequence horizon sequence number

hzn_top horizon top (cm)

hzn_bot horizon bottom (cm)

hzn_desgn horizon name

texture_description USDA soil texture

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

ex_acid exchangeable acidity [method ?]

CEC8.2 cation exchange capacity by sum of cations method (pH 8.2)

CEC7 cation exchange capacity by ammonium acetate (pH 7)

bs_8.2 base saturation by sum of cations method (pH 8.2)

bs_7 base saturation by ammonium acetate (pH 7)

Details

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

Source

<http://ssldata.nrcs.usda.gov/>

Examples

```
data(ca630)
str(ca630)

# 2. promote to SoilProfileCollection
# combine site+horizon data into single DF
ca <- join(ca630$lab, ca630$site, type='inner')

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

# extract site data
site(ca) <- ~ mlra + ssa + lon + lat + cntrl_depth_to_top + cntrl_depth_to_bot + sampled_taxon_name

# extract spatial data as SpatialPoints
coordinates(ca) <- ~ lon + lat
# assign CRS data
proj4string(ca) <- '+proj=latlong +datum=NAD83'

# check the result
ca

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

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

# 4. 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)

xyplot(
  top ~ p.q50, groups=factor(mlra, levels=c('18', '22')), data=a, lower=a$p.q25, upper=a$p.q75,
  ylim=c(160,-5), alpha=0.5, scales=list(y=list(tick.num=7, alternating=3), x=list(alternating=1)),
  panel=panel.depth_function, prepanel=prepanel.depth_function,
  ylab='Depth (cm)', xlab='Base Saturation at pH 8.2',
  par.settings=list(superpose.line=list(col=c('black', 'blue'), lty=c(1,2), lwd=2)),
  auto.key=list(columns=2, title='MLRA', points=FALSE, lines=TRUE)
)

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

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

# note that the ordering is preserved:
all.equal(s.25$pedon_key, profile_id(ca))

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

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

# 8. Extract the all horizons from profiles 1:10 as data.frame
ca.1.to.10 <- ca[1:10, ]

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

```

f.noise

Example Objective Function for Full-Pattern Matching

Description

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

Usage

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

Arguments

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

Details

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

Value

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

Author(s)

Dylan E. Beaudette

References

- Chipera, S.J., & Bish, D.L. (2002) FULLPAT: A full-pattern quantitative analysis program for X-ray powder diffraction using measured and calculated patterns. *J. Applied Crystallography*, 35, 744-749.
- Bish, D. 1994. Quantitative Methods in Soil Mineralogy, in *Quantitative X-Ray Diffraction Analysis of Soil*. Amonette, J. & Zelazny, L. (ed.) Soil Science Society of America, pp 267-295.

See Also

[resample.twotheta](#)

Examples

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

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

# number of components
n.components <- 6

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

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

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

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

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

## fit pattern mixtures with a linear model
l <- lm(synthetic_pat ~ nontronite + montmorillonite + 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)
#

```

generalize.hz	<i>Generalize Horizon Names</i>
---------------	---------------------------------

Description

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

Usage

```
generalize.hz(x, new, pat)
```

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

Value

factor of the same length as x

Author(s)

Dylan E. Beaudette

References

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

Examples

```
data(sp1)

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

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

# see how we did / what we missed
table(sp1$genhz, sp1$name, useNA='always')
```

`munsell`*Munsell-RGB 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 observations on the following 6 variables.

hue Munsell Hue, upper case

value Munsell Value

chroma Munsell Chroma

r red value (0-1)

g green value (0-1)

b blue value (0-1)

Details

See `munsell2rgb` for conversion examples.

Source

Color chip XYZ values: <http://www.cis.rit.edu/mcsl/online/munsell.php>

References

<http://www.brucelindbloom.com/index.html?ColorCalcHelp.html> Color conversion equations

<http://casoilresource.lawr.ucdavis.edu/drupal/node/201> Methods used to generate this table

Examples

```
data(munsell)
```

munsell2rgb	<i>Convert Munsell Notation to RGB</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)
```

Arguments

the_hue	a vector of one or more more hues, upper-case
the_value	a vector of one or more values
the_chroma	a vector of one or more chromas
alpha	alpha channel value (for transparency effects)
maxColorValue	maximum RGB color value (see rgb)
return_triplets	should the function return raw RGB triplets instead of an R color

Details

This function generalizes to vectorized usage, as long as the length of each argument is the same.

Value

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

Warning

As of `plyr` 1.6 (CRAN), there are cases when this function will fail (<https://github.com/hadley/plyr/issues/43>). The next version of `plyr` should address this problem. Until then, be sure not to pass in Munsell (hue, value, chroma) as factors.

Note

Care should be taken when using the resulting RGB 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. Th 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)

Dylan E. Beaudette

References

<http://casoilresource.lawr.ucdavis.edu/drupal/node/201> <http://www.brucelindbloom.com/index.html?ColorCalcHelp.html> <http://www.cis.rit.edu/mcsl/online/munsell.php>

Examples

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

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

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

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

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

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

```

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

}

# soils example
data(sp1)

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

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

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

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

```

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.

Usage

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

Arguments

x	x values (generated by calling lattice function)
y	y values (generated by calling lattice function)

<code>id</code>	vector of id labels, same length as <code>x</code> and <code>y</code> —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 <code>lattice</code> function)
<code>groups</code>	grouping data (generated by calling <code>lattice</code> function)
<code>sync.colors</code>	optionally sync the fill color within the region bounded by (<code>lower</code> – <code>upper</code>) with the line colors
<code>cf</code>	optionally annotate contributing fraction data at regular depth intervals see slab
<code>...</code>	further arguments to lower-level <code>lattice</code> plotting functions, see below

Details

This function can be used to replace `panel.superpose` when plotting depth function data.

Author(s)

Dylan E. Beaudette

References

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

See Also

[sp1](#)

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'))))
)
```

```
# 2. plotting upper/lower envelope around some value, by depth:
# profile data are aggregated ("slotted") by 1cm depth slices
a <- soil.slot(sp1)

# manually add mean +/- SD
a$upper <- with(a, p.mean+p.sd)
a$lower <- with(a, p.mean-p.sd)

# use custom plotting function for uncertainty viz.
xyplot(top ~ p.mean, data=a,
lower=a$lower, upper=a$upper, ylim=c(250,-5), alpha=0.5,
panel=panel.depth_function,
prepanel=prepanel.depth_function
)
```

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)
```

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

Description

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

Value

A vector of length `nrow(object)` (horizon data) or of length `length(object)` (site data).

Methods

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

Examples

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

# split a SPC into a list of single-profile SPC objects
# used internally by profileApply()
str(splitProfiles(sp1), 1)

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

# compute depth-wise differencing by profile
```

```

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

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

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

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

## use the digest library to detect duplicate data
data(sp1)

# make a copy, stack, and give new IDs
s.1 <- sp1
s.2 <- sp1
s.2$id <- paste(s.2$id, '-copy', sep='')
s <- rbind(s.1, s.2)
depths(s) <- id ~ top + bottom
plot(s)

# setup site data, so that we can save md5 hash to @site later
site(s) <- ~ group

# eval dupes with digest, save md5 hash into @site
# note that we are only working with horizon data
# note that we are removing the 1st column, as it contains the profile ID
if(require(digest)) {
s$md5 <- profileApply(s, function(x) digest(unlist(horizons(x)[, -1])))

# get unique hashes
u.md5 <- unique(s$md5)

# list profile idx by hash:
profiles.by.hash <- sapply(u.md5, function(i) which(s$md5 == i), simplify=FALSE)

# get an index of the first copy of each profile
u.profiles <- sapply(profiles.by.hash, function(x) x[1])

# check: OK
plot(s[u.profiles, ])
}

```

```

##
## helper functions: these must be modified to suit your own data
##

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

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

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

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

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

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

```

```

    if(any(is.na(sv)))
      return(NA)

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

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

```

 profile_compare-methods

Numerical Soil Profile Comparison

Description

Performs a numerical comparison of soil profiles using named properties, based on a weighted, summed, depth-segment-aligned dissimilarity calculation. If `s` is a [SoilProfileCollection](#), site-level variables (2 or more) can also be used. The site-level and horizon-level dissimilarity matrices are then re-scaled to [0,1] and averaged.

Usage

```

pc(s, vars, max_d, k, 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)

```

Arguments

<code>s</code>	a dataframe with at least 2 columns of soil properties, and an 'id' column for each profile. horizon depths must be integers and self-consistent
<code>vars</code>	A vector with named properties that will be used in the comparison. These are typically column names describing horizon-level attributes (2 or more), but can also contain site-level attributes (2 or more) if <code>s</code> is a SoilProfileCollection .
<code>max_d</code>	depth-slices up to this depth are considered in the comparison
<code>k</code>	a depth-weighting coefficient, use '0' for no depth-weighting (see examples below)
<code>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 to [0,1] (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.

Value

A dissimilarity matrix object of class 'dist', optionally scaled to [0, 1].

Methods

data = "SoilProfileCollection" see [SoilProfileCollection](#)

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

Note

... based on the work of

Author(s)

Dylan E. Beaudette

References

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

See Also

[unroll](#), [soil.slot](#)

Examples

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

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

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

# better plotting with ape package:
require(ape)
h <- diana(d)
p <- as.phylo(as.hclust(h))
plot(ladderize(p), cex=0.75, label.offset=1, no.margin=TRUE)
tiplabels(col=cutree(h, 3), pch=15)

## 3. other uses of the dissimilarity matrix
require(MASS)
# 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))

## 4. 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)

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

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

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

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

# remove bad profiles and try again: works
dd <- profile_compare(d[-missing.too.much.data.idx, ], vars=c('p1', 'p2'), max_d=100, k=0)

```

random_profile

Random Profile

Description

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

Usage

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

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
...	additional parameters passed-in to the LPP (.lpp) function

Details

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).

Value

A dataframe with the simulated profile.

Note

See examples for ideas on simulating several profiles at once.

Author(s)

Dylan E. Beaudette

References

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

See Also

[profile_compare](#)

Examples

```

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

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

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

# make a more interesting color based on the first property
# depth functions are generated using the LPP function
if(require(scales)) {
  opar <- par(mfrow=c(2,1))

  # setup color palette and mapping function
  cols <- rev(brewer_pal(pal='Spectral')(8))
  grad <- gradient_n_pal(cols)

  # generate data and color scales
  d <- ldply(1:10, random_profile, n=c(6, 7, 8), n_prop=1, method='LPP')
  d$soil_color <- cscale(d$p1, grad)

  # promote to SPC and plot
  depths(d) <- id ~ top + bottom
  plot(d)
  legend('bottom', legend=pretty(d$p1), pt.bg=cscale(pretty(d$p1), grad),
        pch=22, pt.cex=2.5, bty='n', horiz=TRUE)

  # do this again, this time set all of the LPP parameters
  d <- ldply(1:10, random_profile, n=c(6, 7, 8), n_prop=1, method='LPP',
        lpp.a=5, lpp.b=10, lpp.d=5, lpp.e=5, lpp.u=25)
  d$soil_color <- cscale(d$p1, grad)
  depths(d) <- id ~ top + bottom
  plot(d)
  legend('bottom', legend=pretty(d$p1), pt.bg=cscale(pretty(d$p1), grad),
        pch=22, pt.cex=2.5, bty='n', horiz=TRUE)

  par(opar)
}

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

(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)
  h$mid <- with(h, (top + bottom) / 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))))
}

# stress-test profile comparison functions (not run)
# d <- ldply(1:1000, random_profile)
#
# 100 profiles, 4 variables:
# 66 seconds on 1.3 Ghz Intel Mac Mini
# D matrix = 192.3 Mb
# 768 Mb required
# p <- profile_compare(d, vars=c('p1','p2','p3','p4','p5'), max_d=50, k=0)

# more efficient computation, at the expense of precision, with
# p <- profile_compare(d, vars=c('p1','p2','p3','p4','p5'),
# max_d=50, k=0, sample_interval=10)

```

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')
```

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)
```

slice-methods

~~ *Methods for Function slice in Package aqp* ~~

Description

~~ Methods for function slice in package **aqp** ~~

Methods

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

Author(s)

Dylan E. Beaudette

References

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

Examples

```

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

# re-order IDs
new.order <- sample(1:20)
head(d <- d[order(match(d$id, new.order), d$top), ])

# init SoilProfilecollection object: IDs are reset according to natural order of ID labels
depths(d) <- id ~ top + bottom
head(horizons(d))

# check order of IDs: these match the re-ordered horizon data
profile_id(d)

# slice-up at 10 cm: these IDs match the output from profile_id()
# output a data.frame instead of a SoilProfilecollection
s <- slice(d, 10 ~ name + p1 + p2 + p3, just.the.data=TRUE)

# are the results in the same order as our original IDs?
if( ! all.equal(s$id, profile_id(d)))
  stop('IDs do not match')

```

soil.slot

Slice-Wise Aggregation of Soil Properties

Description

Align a single soil property to a user-defined basis, and perform slice-wise aggregation.

Usage

```

soil.slot(data, seg_size = NA, seg_vect = NA,
use.wts = FALSE, strict = FALSE, user.fun = NULL, class_prob_mode=1)

```

Arguments

data	A dataframe representing a 'stack' of soil profiles and having the following format: id An id that is unique across soil profiles within the dataframe. top The horizon top boundary, must be an integer. bottom The horizon bottom boundary, must be an integer. prop A property to be aggregated. can be numeric or a factor.
seg_size	User-defined segment size, default is 1.

<code>seg_vect</code>	User-defined segment structure: should start from 0, and deepest boundary should be deeper than the deepest soil profile in the collection. For example, if the deepest profile in the collection is 200 cm, then the following segmenting vector would be reasonable: <code>c(0, 10, 20, 30, 60, 100, 150, 210)</code> . The resulting aggregation will automatically be truncated at 200 cm. The user is responsible for supplying sensible values.
<code>use.wts</code>	If TRUE, then a column called 'wt' should be present in the source dataframe, and must make sense within the context of the data (i.e. area weights, etc.). Weighted means, standard deviations, quantiles, and optionally proportions will be returned by depth-slice. See details and examples below.
<code>strict</code>	should horizons be strictly checked for self-consistency? defaults to FALSE
<code>user.fun</code>	User-defined function that should accept a vector and return a scalar. This function should know how to properly deal with unexpected, NA, NULL, or Inf values and trap them accordingly.
<code>class_prob_mode</code>	Strategy for normalizing slice-wise probabilities, dividing by either: number of profiles with data at the current slice (<code>class_prob_mode=1</code>), or by the number of profiles in the collection (<code>class_prob_mode=2</code>). Mode 2 values will always sum to the contributing fraction, while mode 1 values will always sum to 1. Mode 2 is likely the best way to communicate horizon probability near the lower range in soil depth within a collection of soil profiles.

Details

Unweighted and weighted summary stats are computed with the Hmisc functions `wtd.mean`, `wtd.var`, and `wtd.quantile`. Weighted probabilities (proportions) will be implemented in a future release. See the sample dataset 'sp1' documentation for further examples on how to use `soil.slot`. 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.

Data are returned according to the following: A dataframe with slice-wise aggregation. When `prop` is numeric a dataframe is returned in the following format:

top The slice top boundary.

bottom The slice bottom boundary.

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

p.mean The slice-wise (optionally weighted) mean.

p.sd An slice-wise (optionally weighted) standard deviation.

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

When `prop` is a factor variable, slice-wise probabilities for each level of `prop`:

top The slice top boundary.

bottom The slice bottom boundary.

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

A The slice-wise probability of level A

B The slice-wise probability of level B

...

n The slice-wise probability of level n

Value

See Details section above.

Warning

These examples are now obsolete, use with cation.

Note

If a user-defined function (`user.fun`) is specified, care must be taken if sample size is used within the calculation `_and_` slice sizes are > 1 depth unit.

Note

This function is used internally by other functions. Examples below should be used with caution, as they will be migrated into higher-level documentation soon.

Author(s)

Dylan E Beaudette

References

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

See Also

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

Examples

```
# load example data
data(sp1)

# test that mean of 1cm slotted property is equal to the
# hz-thickness weighted mean value of that property
sp1.sub <- subset(sp1, sub=id == 'P009')
hz.wt.mean <- with(sp1.sub,
  sum((bottom - top) * prop) / sum(bottom - top))
```

```

)
# hopefully the same value, calculated with soil.slot()
a <- soil.slot(sp1.sub)
# same?
if(!all.equal(mean(a$p.mean), hz.wt.mean))
stop('there is a bug in soil.slot() !!!')

# calculate a weighted average of some property over a slab of soil
s <- c(100,150)
soil.slot(sp1.sub, seg_vect=s)$p.mean

#
# 1 standard usage, and plotting example
#

# slot at two different segment sizes
a <- soil.slot(sp1)
b <- soil.slot(sp1, seg_size=5)

# 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'))

# manually add mean +/- SD
ab$upper <- with(ab, p.mean+p.sd)
ab$lower <- with(ab, p.mean-p.sd)

# use mean +/- 1 SD
# custom plotting function for uncertainty viz.
xyplot(top ~ p.mean | which, data=ab, ylab='Depth',
xlab='mean bounded by +/- 1 SD',
lower=ab$lower, upper=ab$upper, ylim=c(250,-5), alpha=0.5,
panel=panel.depth_function,
prepanel=prepanel.depth_function,
layout=c(2,1), scales=list(x=list(alternating=1))
)

# use 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), alpha=0.5,
panel=panel.depth_function,
prepanel=prepanel.depth_function,
layout=c(2,1), scales=list(x=list(alternating=1))
)

```

```

#
# 1.1 try slotting categorical variables
#

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

# generate new data for testing soil.slot()
y <- with(sp1, data.frame(id=id, top=top, bottom=bottom, prop=name))
# convert name to a factor
y$prop <- factor(y$prop)
# fix factor levels
y$id <- factor(y$id)

# default slotting-- 1cm intervals,
# adjusting slice-wise probability with contributing fraction
a <- soil.slot(y, class_prob_mode=1)

# reshape into long format for plotting
a.long <- melt(a, id.var=c('top','bottom'))

a.long$variable <- factor(a.long$variable, levels=c('O','A','B','C','R'))

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

## ajust probability to size of collection
a.1 <- soil.slot(y, class_prob_mode=2)

# reshape into long format for plotting
a.1.long <- melt(a.1, id.var=c('top','bottom'))

# group mode 1 and mode 2 data
g <- make.groups(mode_1=a.long, mode_2=a.1.long)
g$variable <- factor(g$variable, levels=c('O','A','B','C','R'))

# plot horizon type proportions
xyplot(top ~ value | variable, groups=which, data=g, subset=value > 0,
ylim=c(150, -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))))

## compare class probability values when changing segment size
a.5 <- soil.slot(y, class_prob_mode=2, seg_size=5)

# reshape into long format for plotting
a.5.long <- melt(a.5, id.var=c('top','bottom'))

```

```

# group 1cm and 5cm slices
g <- make.groups(s1cm=a.1.long, s5cm=a.5.long)
g$variable <- factor(g$variable, levels=c('0','A','B','C','R'))

xyplot(top ~ value | variable, groups=which, data=g, subset=value > 0,
ylim=c(150, -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))))

#
# 2. depth probability via contributing fraction
# note that this assumes that we are not missing data in 'prop'
# get around NA by making a fake column filled with 1
# like this:
# sp1$prop <- 1
#
a <- soil.slot(sp1)
xyplot(top ~ contributing_fraction, data=a,
ylim=c(250, -5), type='S', horizontal=TRUE, asp=4)

#
# 3.1 standard aggregation
#
a <- soil.slot(sp1)

# manually add mean +/- SD
a$upper <- with(a, p.mean+p.sd)
a$lower <- with(a, p.mean-p.sd)

# use custom plotting function for uncertainty viz.
xyplot(top ~ p.mean, data=a,
lower=a$lower, upper=a$upper, ylim=c(250,-5), alpha=0.5,
panel=panel.depth_function,
prepanel=prepanel.depth_function
)

#
# 3.3 use of weights
#
data(sp1)

# some fake weights
wts <- data.frame(id=unique(sp1$id), wt=c(3,1,2,1,2,1,4,1,2))

# merge with original data
g <- merge(sp1, wts, by='id')

# generate horizon mid points

```

```

g$mid <- with(g, (bottom + top) / 2)

# aggregate and add upper/lower intervals via SD
a <- soil.slot(g, use.wts=TRUE)
a$upper <- with(a, p.mean + p.sd)
a$lower <- with(a, p.mean - p.sd)

a$wt.upper <- with(a, p.wtmean + p.wtsd)
a$wt.lower <- with(a, p.wtmean - p.wtsd)

# check influence of weights
plot(mid ~ prop, data=g, ylim=c(240,0), cex=sqrt(g$wt), xlim=c(-5,60))
lines(top ~ p.mean, data=a, lwd=2)
lines(top ~ upper, data=a, lty=2)
lines(top ~ lower, data=a, lty=2)

lines(top ~ p.wtmean, data=a, col=2, lwd=2)
lines(top ~ wt.upper, data=a, col=2, lty=2)
lines(top ~ wt.lower, data=a, col=2, lty=2)

# annotate with explanation
legend('bottomright',
legend=c('wt = 1', 'wt = 2', 'wt = 3', 'wt = 4', 'un-weighted', 'weighted'),
pch=c(1,1,1,1,NA,NA), lwd=c(1,1,1,1,2,2), lty=c(NA,NA,NA,NA,1,1),
pt.cex=sqrt(c(1,2,3,4,1,1)), col=c(1,1,1,1,1,2), bty='n')

#
# try again with larger segment sizes
#

# aggregate and add upper/lower intervals via SD
a <- soil.slot(g, use.wts=TRUE, seg_size=10)

a$upper <- with(a, p.mean + p.sd)
a$lower <- with(a, p.mean - p.sd)
a$wt.upper <- with(a, p.wtmean + p.wtsd)
a$wt.lower <- with(a, p.wtmean - p.wtsd)

# convert to long format
library(reshape)
a.long <- melt(a, id.var=c('top', 'bottom'),
measure.var=c('p.mean', 'p.wtmean', 'upper', 'lower', 'wt.upper', 'wt.lower'))

# red lines are weighted
# point symbols are sized proportional to their weights
xyplot(cbind(top, bottom) ~ value, groups=variable, data=a.long, id=g$id,
ylim=c(260,-10), ylab='Depth', xlab='Property',
par.settings=list(superpose.line=list(
col=c('black', 'red', 'black', 'black', 'red', 'red'),
lwd=c(2,2,1,1,1,1),

```

```

lty=c(1,1,2,2,2,2)),
panel=function(...) {
panel.points(g$prop, g$mid, cex=sqrt(g$wt), col=1)
panel.depth_function(...)
}
)

```

SoilProfileCollection-class

SoilProfileCollection Class

Description

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

Objects from the Class

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

Slots

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

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

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

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

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

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

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

Methods

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

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

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

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

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

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

```

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

```

Author(s)

Pierre Roudier and Dylan E. Beaudette

Examples

```

## 1. test validity methods

# this should work fine
data(sp1)
depths(sp1) <- id ~ top + bottom

# horizon logic can be tested via data.frame and
# test_hz_logic(i, topcol, bottomcol, test.NA=TRUE, strict=FALSE)

## Not run:
# these next examples should throw an error
# insert a missing horizon boundary
data(sp1)
sp1$top[1] <- NA
depths(sp1) <- id ~ top + bottom

# insert a bogus horizon boundary
## NOTE: this is currently valid, as this check breaks slice(SPC, ...)
data(sp1)
sp1$top[2] <- 30

```

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

## End(Not run)
```

SoilProfileCollection-plotting-methods
Profile Plot

Description

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

Usage

```
plotSPC(x, color='soil_color', width=0.2, name='name',
  cex.names=0.5, cex.depth.axis=cex.names, cex.id=cex.names+(0.2*cex.names), print.id=TRUE,
  id.style='auto', plot.order=1:length(x), add=FALSE, scaling.factor=1, y.offset=0, n=length(x),
  max.depth=max(x), n.depth.ticks=5, shrink=FALSE, shrink.cutoff=3, abbr=FALSE,
  abbr.cutoff=5, divide.hz=TRUE, ...)
```

Arguments

<code>x</code>	a SoilProfileCollection object
<code>color</code>	the name of the column containing R-compatible color descriptions
<code>width</code>	scaling of profile widths
<code>name</code>	the name of the column containing the horizon designation
<code>cex.names</code>	character scaling applied to horizon names
<code>cex.depth.axis</code>	character scaling applied to depth scale
<code>cex.id</code>	character scaling applied to profile id
<code>print.id</code>	should the profile id be printed above each profile? (TRUE)
<code>id.style</code>	profile ID printing style: 'auto' (default) = simple heuristic used to select from: 'top' = centered above each profile, 'side' = 'along the top-left edge of profiles'
<code>plot.order</code>	a vector describing the order in which individual SoilProfile objects from the parent should be plotted
<code>add</code>	add to an existing figure
<code>scaling.factor</code>	vertical scaling of the profile heights
<code>y.offset</code>	vertical offset for top of profiles
<code>n</code>	integer describing amount of space along x-axis to allocate, defaults to length(x)
<code>max.depth</code>	suggested lower depth boundary of plot
<code>n.depth.ticks</code>	suggested number of ticks in depth scale
<code>shrink</code>	should long horizon names be shrunk by 80% ?
<code>shrink.cutoff</code>	character length defining long horizon names

abbr	should the profile ID be abbreviated?
abbr.cutoff	suggested minimum length for abbreviated IDs
divide.hz	should horizons be divided with a thin black line? (default is TRUE)
...	other arguments passed into lower level plotting functions

Details

Depth limits (`max.depth`) and number of depth ticks (`n.depth.ticks`) are *suggestions* to the `pretty()` function. You may have to tinker with both parameters to get what you want. The `'side'` `id.style` is useful when plotting a large collection of profiles, and/or, when profile IDs are long.

Value

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

Methods

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

Author(s)

Dylan E. Beaudette

References

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

See Also

[SoilProfileCollection-class](#), [pretty](#)

Examples

```
data(sp1)

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

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

# plot profiles
plot(sp1, 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)

# plot profiles, using alternate profile ID label style
```

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

# plot horizon color according to some property
# RColorBrewer helps with nice colors
if(require(RColorBrewer)) {
  data(sp3)

  # setup colors
  cols <- rev(brewer.pal(8, 'Spectral'))
  cr <- colorRamp(cols)

  # assign color based on clay content, rescaled to {0,1}
  sp3$soil_color <- rgb(cr(rescaler(sp3$clay, type='range')), max=255)
  depths(sp3) <- id ~ top + bottom
  plot(sp3)
}
```

sp1

Soil Profile Data Example 1

Description

Soil profile data from Pinnacles National Monument, CA.

Usage

```
data(sp1)
```

Format

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

group a numeric vector
id a character vector
top a numeric vector
bottom a numeric vector
bound_distinct a character vector
bound_topography a character vector
name a character vector
texture a character vector
prop a numeric vector
structure_grade a character vector
structure_size a character vector

structure_type a character vector
 stickiness a character vector
 plasticity a character vector
 field_ph a numeric vector
 hue a character vector
 value a numeric vector
 chroma a numeric vector

References

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

Examples

```

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

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

# extract 1-unit thick slices at defined depths
s <- slice(sp1, 0:25 ~ prop + name + soil_color)
plot(s)

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

# check mean +/- 1SD
xyplot(top ~ p.mean + I(p.mean + p.sd) + I(p.mean - p.sd),
data=s1, type='S', horizontal=TRUE, col=1, lty=c(1,2,2),
panel=panel.superpose, ylim=c(110,-5))

# check median & IQR
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))

# other segment sizes
# 5cm
s5 <- slab(sp1, fm= ~ prop, seg_size=5)

# 10cm segments:
s10 <- slab(sp1, fm= ~ prop, seg_size=10)

# 20cm
s20 <- slab(sp1, fm= ~ prop, seg_size=20)

```

```

# variation in segment-weighted mean property: very little
sapply(
  list(s1,s5,s10,s20),
  function(i) {
    with(i, sum((bottom - top) * p.mean) / sum(bottom - top))
  }
)

# combined viz
g2 <- make.groups("1cm interval"=s1, "5cm interval"=s5,
  "10cm interval"=s10, "20cm interval"=s20)

# note special syntax for plotting step function
xyplot(cbind(top,bottom) ~ p.mean, groups=which, data=g2, id=g2$which,
  panel=panel.depth_function, ylim=c(250,-10),
  scales=list(y=list(tick.number=10)), xlab='Property',
  ylab='Depth (cm)', main='Soil Profile Averaging by Slotting',
  auto.key=list(columns=4, points=FALSE, lines=TRUE)
)

```

sp2

Honcut Creek Soil Profile Data

Description

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

Usage

```
data(sp2)
```

Format

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

```

id profile id
surface dated surface
top horizon top in cm
bottom horizon bottom in cm
bound_distinct horizon lower boundary distinctness class
bound_topography horizon lower boundary topography class
name horizon name
texture USDA soil texture class

```

prop field-estimated clay content
 structure_grade soil structure grade
 structure_size soil structure size
 structure_type soil structure type
 stickiness stickiness
 plasticity plasticity
 field_ph field-measured pH
 hue Munsell hue
 value Munsell value
 chroma Munsell chroma
 r RGB red component
 g RGB green component
 b RGB blue component
 soil_color R-friendly encoding of soil color

Author(s)

Dylan E. Beaudette

Source

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

References

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

Examples

```

data(sp2)

# convert into SoilProfileCollection and plot
# plotting order, based on the dated surface each soil was described on
p.surface <- as.numeric(aggregate(sp2$surface, by=list(sp2$id), unique)$x)
p.order <- order(p.surface)

depths(sp2) <- id ~ top + bottom
plot(sp2, plot.order=p.order)

# truncate plot to 200 cm depth, that looks better
plot(sp2, plot.order=p.order, max.depth=200)

# look at numerical distances between profiles
data(sp2)

```

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

# better plotting with ape package:
require(ape)
require(cluster)
h <- diana(d)
p <- as.phylo(as.hclust(h))
plot(ladderize(p), cex=0.75, label.offset=0.25)

# add in the dated surface type via color
tiplabels(col=p.surface, pch=15)

# based on distance matrix values, YMMV
legend(x=0, y=6.1, legend=levels(sp2$surface), col=1:6, pch=15, bty='n', bg='white')
```

sp3

Soil Profile Data Example 3

Description

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

Usage

```
data(sp3)
```

Format

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

```
id soil id
top horizon upper boundary (cm)
bottom horizon lower boundary (cm)
clay clay content
cec CEC by amonium acetate at pH 7
ph pH in 1:1 water-soil mixture
tc total carbon percent
hue Munsell hue (dry)
value Munsell value (dry)
chroma Munsell chroma (dry)
mid horizon midpoint (cm)
ln_tc natural log of total carbon percent
```

L color: l-coordinate, CIE-LAB colorspace (dry)
 A color: a-coordinate, CIE-LAB colorspace (dry)
 B color: b-coordinate, CIE-LAB colorspace (dry)
name horizon name
soil_color horizon color

Details

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

References

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

Examples

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

# required packages
library(ape)
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)))

# make a fake grouping variable, and aggregate all profiles
sp3$group <- factor('A')
a <- slab(sp3, fm=group ~ clay + cec + ph + h + s + v, seg_size=10)

# convert back to wide format, note that aggregation metric affects the result
a.wide.q25 <- cast(a, top + bottom ~ variable, value=c('p.q25'))
a.wide.q50 <- cast(a, top + bottom ~ variable, value=c('p.q50'))
a.wide.q75 <- cast(a, top + bottom ~ variable, value=c('p.q75'))

# add a new id for the 25th, 50th, and 75th percentile pedons
# and convert top/bottoms to integers
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')
sp3.grouped <- rbind(
  sp3[, vars], a.wide.q25[, vars], a.wide.q50[, vars], a.wide.q75[, vars]
)

# re-constitute the soil color from HSV triplets
```

```

# convert HSV back to standard R colors
sp3.grouped$soil_color <- with(sp3.grouped, hsv(h, s, v))

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

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

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

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

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

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

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

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

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

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

```

Description

Soil Chemical Data from Serpentinic Soils of California

Usage

```
data(sp4)
```

Format

A data frame with 30 observations on the following 19 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 Serpentinite Landscapes. *Soil Sci. Soc. Am. J.* 73: 2087-2095.

Examples

```
data(sp4)

sp4$soil_color <- 'white'

# optionally color horizons with 'ex_Ca_to_Mg'
if(require(RColorBrewer)) {
```

```

cols <- rev(brewer.pal(8, 'Spectral'))
cr <- colorRamp(cols)
sp4$soil_color <- rgb(cr(rescaler(sp4$ex_Ca_to_Mg, type='range')), max=255)
}

# init SPC object
depths(sp4) <- id ~ top + bottom

# plot
plot(sp4)

```

sp5

Sample Soil Database #5

Description

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

Usage

```
data(sp5)
```

Format

```

Formal class 'SoilProfileCollection' [package "aqp"] with 6 slots
..@ idcol      : chr "soil"
..@ depthcols: chr [1:2] "top" "bottom"
..@ metadata  :'data.frame': 1 obs. of 1 variable:
.. ..$ depth_units: chr "cm"
..@ horizons  :'data.frame': 1539 obs. of 17 variables:
.. ..$ soil      : soil ID
.. ..$ sand      : sand
.. ..$ silt      : silt
.. ..$ clay      : clay
.. ..$ R25       : RGB r-coordinate
.. ..$ G25       : RGB g-coordinate
.. ..$ B25       : RGB b-coordinate
.. ..$ pH        : pH
.. ..$ EC        : EC
.. ..$ 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://eussoils.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

```

data(sp5)

# 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)

# more fun demos, these require the 'scales' package
if(require(scales)) {

# 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')

# plotting parameters
yo <- 100 # y-offset
sf <- 0.65 # scaling factor
# plot profile sketches
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
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)
}
```

spatial_subset-methods

Spatial Subsetting of SoilProfileCollection Objects

Description

Spatial Subsetting of SoilProfileCollection Objects

Methods

signature(object = "SoilProfileCollection")

Note

This functionality requires the ‘rgeos’ package.

Author(s)

Dylan E. Beaudette

References

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

Description

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

Arguments

<code>data</code>	a dataframe with columns (‘id’, ‘top’, ‘bottom’)
<code>fm</code>	a formula in the form: <code>groups ~ var1 + var2 + var3 + ...</code> OR <code>~ var1 + var2 + var3 + ...</code>
<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>...</code>	further arguments passed onto soil.slot

Details

This function is a wrapper to [soil.slot](#), and allows for slab-wise aggregation of several soil properties at once. The results are in long format; see examples for further processing steps that may be required.

Value

A dataframe similar to the output from [soil.slot](#), but with depth-slice information repeated for each requested input variable and group. If a grouping variable is not given, then all of the profiles in the collection are aggregated into a single representative profile.

Methods

`data = "SoilProfileCollection"` see [SoilProfileCollection](#)

`data = "data.frame"` see [soil.slot](#)

Note

This function has replaced the previously defined ‘`soil.slot.multiple`’ function as of aqp 0.98-8.58.

Author(s)

Dylan E Beaudette

References

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

See Also

[soil.slot](#), [unroll](#)

Examples

```

data(sp3)

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

# slot several variables at once
# within each level of 'group'
# using default parameters to soil.slot()
a <- slab(sp3, fm=group ~ L + A + B)

# pre-compute intervals
a$upper <- with(a, p.mean + p.sd)
a$lower <- with(a, p.mean - p.sd)

# check the results:
# note that 'group' is the column containing group labels
xyplot(
  top ~ p.mean | variable, data=a, groups=group, subscripts=TRUE,
  lower=a$lower, upper=a$upper, ylim=c(125,-5), alpha=0.5,
  layout=c(3,1), scales=list(x=list(relation='free')),
  panel=panel.depth_function,
  prepanel=prepanel.depth_function
)

# convert mean value for each variable into long format
a.wide <- cast(a, group + top + bottom ~ variable, value=c('p.mean'))

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

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

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

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

```

```

## this will work in the next version of plyr
## Not run: library(doSMP)

# setup global option
options('AQP_parallel'=TRUE)

# register workers (2 cores)
w <- startWorkers(workerCount=2, FORCE=TRUE)
registerDoSMP(w)

# run in parallel
a <- slab(sp3, fm=group ~ L + A + B, seg_size=5)

# stop workers (critical!)
stopWorkers(w)

## End(Not run)

```

SPC-utils

Getters, Setters, and Utility Methods for SoilProfileCollection Objects

Description

Getters, Setters, and Utility Methods for SoilProfileCollection Objects

Methods

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

Author(s)

Dylan E. Beaudette

References

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

Examples

```

data(sp1)

## init SoilProfileCollection objects from data.frame
depths(sp1) <- id ~ top + bottom

## depth units
(du <- depth_units(sp1))
depth_units(sp1) <- 'in'
depth_units(sp1) <- du

## get/set metadata on SoilProfileCollection objects

```

```
# this is a 1-row data.frame
m <- metadata(sp1)
m$sampler <- 'Dylan'
metadata(sp1) <- m

## extract horizon data from SoilProfileCollection objects as data.frame
h <- horizons(sp1)

# also replace horizon data in SoilProfileCollection objects
# original order and length must be preserved!
horizons(sp1) <- h

# get number of horizons
nrow(sp1)

## getting site-level data
site(sp1)

## setting site-level data
# site-level data from horizon-level data (stored in @horizons)
site(sp1) <- ~ group

# make some fake site data, and append from data.frame
# a matching ID column must be present in both @site and new data
# note that IDs should all be character class
d <- data.frame(id=profile_id(sp1), p=runif(n=length(sp1)), stringsAsFactors=FALSE)
site(sp1) <- d
```

test_hz_logic

Test Horizon Logic

Description

Simple tests for horizon logic, based on a simple data.frame of ordered horizons.

Usage

```
test_hz_logic(i, topcol, bottomcol, test.NA = TRUE, strict = FALSE)
```

Arguments

i	a data.frame associated with a single soil profile, ordered by depth
topcol	character, giving the name of the column in i that describes horizon top depth
bottomcol	character, giving the name of the column in i that describes horizon bottom depth

test.NA	logical, should tests for missing (NA) horizon top OR bottom boundaries be performed?
strict	logical, should continuity tests be performed– i.e. for non-contiguous horizon boundaries

Details

This function is used as part of the validity-methods for SoilProfileCollection objects.

Value

logical: TRUE -> pass, FALSE -> fail

Author(s)

Dylan E. Beaudette

References

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

See Also

[depths<-](#)

Examples

```
# simple example: just one profile
data(sp1)
depths(sp1) <- id ~ top + bottom
s <- horizons(sp1[1, ])

# apply tests
# passes:
test_hz_logic(s, 'top', 'bottom', test.NA=TRUE, strict=FALSE)
# passes:
test_hz_logic(s, 'top', 'bottom', test.NA=TRUE, strict=TRUE)

# add-in some bad data
s$bottom[6] <- NA # missing horizon boundary, common on bottom-most hz
s$bottom[3] <- 30 # inconsistent hz boundary

# apply tests
# fails due to missing hz boundary
test_hz_logic(s, 'top', 'bottom', test.NA=TRUE, strict=FALSE)
# fails due to inconsistent hz boundary
test_hz_logic(s, 'top', 'bottom', test.NA=FALSE, strict=TRUE)
```

unroll	<i>Unroll Genetic Horizons</i>
--------	--------------------------------

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/>

See Also

[soil.slot](#)

Examples

```
data(sp1)

# subset a single soil profile:
sp1.1 <- subset(sp1, subset=id == 'P001')

# demonstrate how this function works
x <- with(sp1.1, unroll(top, bottom, prop, max_depth=50))
plot(x, 1:length(x), ylim=c(90,0), type='b', cex=0.5)
```

Index

- *Topic **classes**
 - SoilProfileCollection-class, [35](#)
- *Topic **datasets**
 - ca630, [3](#)
 - munsell, [10](#)
 - rruff.sample, [26](#)
 - sp1, [39](#)
 - sp2, [41](#)
 - sp3, [43](#)
 - sp4, [45](#)
 - sp5, [47](#)
- *Topic **hplot**
 - panel.depth_function, [13](#)
 - plot_distance_graph, [15](#)
- *Topic **manip**
 - f.noise, [5](#)
 - generalize.hz, [9](#)
 - munsell2rgb, [11](#)
 - profile_compare-methods, [19](#)
 - profileApply-methods, [16](#)
 - random_profile, [22](#)
 - resample.twotheta, [25](#)
 - slice-methods, [27](#)
 - soil.slot, [28](#)
 - spatial_subset-methods, [50](#)
 - SPC-slab-methods, [51](#)
 - test_hz_logic, [54](#)
 - unroll, [56](#)
- *Topic **methods**
 - profile_compare-methods, [19](#)
 - profileApply-methods, [16](#)
 - slice-methods, [27](#)
 - spatial_subset-methods, [50](#)
 - SPC-slab-methods, [51](#)
 - SPC-utils, [53](#)
- *Topic **package**
 - aqp-package, [2](#)
 - .lpp (random_profile), [22](#)
 - .make.segments (panel.depth_function), [13](#)
 - [,SoilProfileCollection-method (SoilProfileCollection-class), [35](#)
 - \$,SoilProfileCollection-method (SoilProfileCollection-class), [35](#)
 - \$<-,SoilProfileCollection-method (SoilProfileCollection-class), [35](#)
 - aqp (aqp-package), [2](#)
 - aqp-package, [2](#)
 - ca630, [3](#), [3](#)
 - coordinates,SoilProfileCollection-method (SoilProfileCollection-class), [35](#)
 - coordinates<-,SoilProfileCollection-method (SoilProfileCollection-class), [35](#)
 - create_progress_bar, [20](#), [51](#)
 - depth_units (SPC-utils), [53](#)
 - depth_units,SoilProfileCollection-method (SPC-utils), [53](#)
 - depth_units<- (SPC-utils), [53](#)
 - depth_units<-,SoilProfileCollection-method (SPC-utils), [53](#)
 - depths<-, [55](#)
 - depths<- (SPC-utils), [53](#)
 - depths<-,data.frame-method (SPC-utils), [53](#)
 - diagnostic_hz (SPC-utils), [53](#)
 - diagnostic_hz,SoilProfileCollection-method (SPC-utils), [53](#)
 - diagnostic_hz<- (SPC-utils), [53](#)
 - diagnostic_hz<-,SoilProfileCollection-method (SPC-utils), [53](#)
 - f.noise, [5](#)

- generalize.hz, 9
- get.slice (slice-methods), 27
- horizonDepths (SPC-utils), 53
- horizonDepths, SoilProfileCollection-method (SPC-utils), 53
- horizons (SPC-utils), 53
- horizons, SoilProfileCollection-method (SPC-utils), 53
- horizons<- (SPC-utils), 53
- horizons<-, SoilProfileCollection-method (SPC-utils), 53
- idname (SPC-utils), 53
- idname, SoilProfileCollection-method (SPC-utils), 53
- length, SoilProfileCollection-method (SoilProfileCollection-class), 35
- max, SoilProfileCollection-method (SoilProfileCollection-class), 35
- metadata (SPC-utils), 53
- metadata, SoilProfileCollection-method (SPC-utils), 53
- metadata<- (SPC-utils), 53
- metadata<-, SoilProfileCollection-method (SPC-utils), 53
- min, SoilProfileCollection-method (SoilProfileCollection-class), 35
- munsell, 10
- munsell2rgb, 11
- names, SoilProfileCollection-method (SoilProfileCollection-class), 35
- nrow, SoilProfileCollection-method (SoilProfileCollection-class), 35
- panel.depth_function, 13
- pc (profile_compare-methods), 19
- plot, SoilProfileCollection-method (SoilProfileCollection-plotting-methods), 37
- plot_distance_graph, 15
- plotSPC (SoilProfileCollection-plotting-methods), 37
- prepanel.depth_function (panel.depth_function), 13
- pretty, 38
- profile_compare, 16, 20, 23
- profile_compare (profile_compare-methods), 19
- profile_compare, data.frame-method (profile_compare-methods), 19
- profile_compare, SoilProfileCollection-method (profile_compare-methods), 19
- profile_compare-methods, 19
- profile_id (SPC-utils), 53
- profile_id, SoilProfileCollection-method (SPC-utils), 53
- profileApply (profileApply-methods), 16
- profileApply, SoilProfileCollection-method (profileApply-methods), 16
- profileApply-methods, 16
- proj4string, SoilProfileCollection-method (SoilProfileCollection-class), 35
- proj4string<-, SoilProfileCollection-method (SoilProfileCollection-class), 35
- random_profile, 22
- resample.twotheta, 6, 25
- rgb, 11
- rruff.sample, 26, 26
- seg.summary (soil.slot), 28
- show, SoilProfileCollection-method (SoilProfileCollection-class), 35
- site (SPC-utils), 53
- site, SoilProfileCollection-method (SPC-utils), 53
- site<- (SPC-utils), 53
- site<-, SoilProfileCollection-method (SPC-utils), 53
- slab, 14, 30
- slab (SPC-slab-methods), 51
- slab, data.frame-method (SPC-slab-methods), 51
- slab, SoilProfileCollection-method (SPC-slab-methods), 51

slice (slice-methods), [27](#)
slice, SoilProfileCollection-method
 (slice-methods), [27](#)
slice-methods, [27](#)
slice.fast (slice-methods), [27](#)
soil.slot, [21](#), [28](#), [51](#), [52](#), [56](#)
SoilProfileCollection, [19](#), [20](#), [51](#)
SoilProfileCollection
 (SoilProfileCollection-class),
 [35](#)
SoilProfileCollection-class, [38](#)
SoilProfileCollection-class, [35](#)
SoilProfileCollection-plotting-methods,
 [37](#)
sp1, [3](#), [14](#), [30](#), [39](#)
sp2, [3](#), [16](#), [41](#)
sp3, [3](#), [43](#)
sp4, [3](#), [45](#)
sp5, [3](#), [47](#)
spatial_subset
 (spatial_subset-methods), [50](#)
spatial_subset, SoilProfileCollection-method
 (spatial_subset-methods), [50](#)
spatial_subset-methods, [50](#)
SPC-slab-methods, [51](#)
SPC-utils, [53](#)
splitProfiles (profileApply-methods), [16](#)
splitProfiles, SoilProfileCollection-method
 (profileApply-methods), [16](#)

test_hz_logic, [54](#)

unroll, [21](#), [30](#), [52](#), [56](#)