# Soil Database Interface
The `soilDB` package

D.E. Beaudette and J.M. Skovlin

February 27, 2012

## 1 Introduction

This package provides methods for extracting soils information from local PedonPC and AK Site databases (MS Access format), local NASIS databases (MS SQL Server), and the SDA webservice. Currently USDA-NCSS data sources are supported, however, there are plans to develop interfaces to outside systems such as the Global Soil Mapping project.

It can be difficult to locate all of the dependencies required for sending/processing SOAP requests, especially on UNIX-like operating systems. Windows binary packages for the dependencies can be found here `http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/2.14/`.

### 1.1 Inteded Users

### 1.2 R Notes

A basic understanding of R syntax, object structure, package management system, and programming style will greatly help with integration of the `soilDB` `aqp` into standard workflows. Here are some reminders. Recall that online help documents can be accessed with the syntax '?mean', which would load the manual page for the 'mean()' function. Two '??' can be used to perform fuzzy searching for functions that contain matching keywords.

```r
# 1. objects: creation, structure, etc.
x <- 1:10  # 'x' is now a vector of integers 1 to 10
y <- rnorm(10)  # 'y' is now a vector of 10 random numbers {mean=0, sd=1}
d <- data.frame(x, y)  # d is a rectangular table with columns 'x' and 'y'

str(d)  # check structure
class(d)  # inspect object class
head(d)  # view first 6 rows
rm(d, x, y)  # clean-up by deleting these objects

# 2. most function are vectorized: i.e. automatic
#   iteration
x <- 1:10
x + 1  # the '+' function knows about vectors, and recycles scalars accordingly

# 3. missing data can cause problems unless accounted
#   for
x <- c(x, NA)  # 'c()' is the concatonate function
x
mean(x)  # result is NA, as most functions return NA in the presence of missing data
mean(x, na.rm = TRUE)  # mean computed after removal of missing data
```

### 1.3 Design

The `soilDB` package provides two layers of functionality for extracting data from file-based databases (PedonPC), relational databases (local NASIS), and online data sources (Soil Data Access). High-level functions `fetchPedonPC()` and `fetchNASIS()`

query, combine, verify horizon logic, and return `SoilProfileCollection` class objects containing the most commonly used site/pedon/horizon data. Low-level functions such as `get_site_data_from_pedon_db()` and `get_hz_data_from_pedon_db()` extract specific data from a designated source, and return the results as a `data.frame` class object.

## 1.4 High-Level Functions

## 1.5 Low-Level Functions

```r
library(soilDB)

## setup: this will only work if you have a PedonPC
#   Database and Windows :( ##

# get all of the data (you will have to adjust the DSN)
dsn <- "S:/Service_Center/NRCS/pedon/pedon.mdb"
s <- fetchPedonPC(dsn)
e <- get_extended_data_from_pedon_db(dsn)

# subset Gopheridge pedons
subset.idx <- grep("gopheridge", s$sampled_as,
    ignore.case = TRUE)
s <- s[subset.idx, ]

# move peiid into @site, so that we can use as join
#   condition for diagnostic data
site(s) <- ~peiid


## diagnostic features ##

# extract diagnostic hz data (all records)
d <- e$diagnostic

# save only those rows with paralithic material/contact
s.paralithic <- d[grep("paralithic", d$diag_kind,
    ignore.case = TRUE), ]


## adding data to @site slot ##

# save to @site: via LEFT join by peiid
# this will ignore all pedons not already in 's'
# note that other diagnostic data could be added to
#   @site, by re-naming the columns in s.paralithic
# names(s.paralithic) <- c('peiid', 'para', 'para.top',
#   'para.bottom')
site(s) <- s.paralithic
```

### 1.6 Possible Applications

## 2 Linkages to the 'aqp' Package

### 2.1 The 'SoilProfileCollection' Class and Methods

## 3 Sample Dataset: 'loafercreek'

## 4 Sample Dataset: 'gopheridge'

## 5 Examples

```r
# libraries: you may have to install these
library(soilDB)
library(ape)

# load stored SoilProfileCollection
data(gopheridge)

# temp hack: save a copy in 's'
s <- gopheridge

## aggregates of horizon data, moved into @site

# horizon-thickness weighted mean (beware of NA)
f.wt.prop <- function(x, prop) {
    h <- horizons(x)
    w <- with(h, hzdepb - hzdept)
    wtd.mean(h[[prop]], weights = w)
}

# apply by-profile, returning a vector, assigned to new
#   column in @site
s$wt.clay <- profileApply(s, f.wt.prop, prop = "clay")
s$wt.rf <- profileApply(s, f.wt.prop, prop = "total_frags_pct")

# soil depth using an in-line function, note that
#   max(SPC) returns the bottom-most depth
s$soil.depth <- profileApply(s, function(x) max(x))

# compute aggregate (wt.mean) clay within particle-size
#   control section
# note that this requires conditional eval of data that
#   may contain NA
# see ?slab and ?soil.slot for details on the syntax
f.pcs.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)

    # 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)
}


# 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)
}



# compute wt.mean clay within PCS, save to @site
```

```
s$pcs.clay <- profileApply(s, f.pcs.prop, prop = "clay")

# order by presence of paralithic materials/contact
new.order <- order(s$diag_kind)
plot(s, name = "hzname", plot.order = new.order)
# annotate paralithic data with lines
x.pos <- 1:length(s)
lines(x.pos, s$featdept[new.order], lty = 2, col = "black")
lines(x.pos, s$featdepb[new.order], lty = 2, col = "black")
```

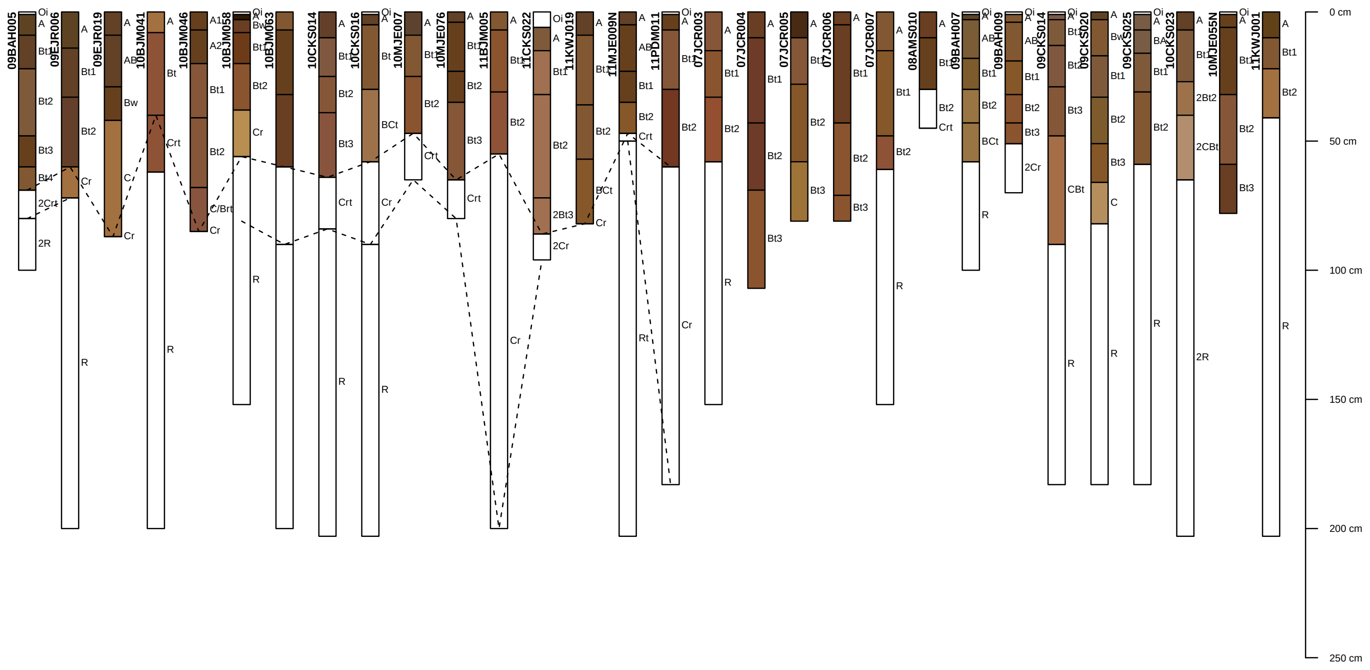

Figure 1: Basic plotting

```
## assess dissimilarity: only horizon attributes for
#   now ##

# D evaluated to 100 cm depth, using clay, sand, total
#   RF, no depth-weighting
d <- profile_compare(s, vars = c("clay", "sand",
    "total_frags_pct"), k = 0, max_d = 100)

# divising hierarchical clustering
h.hclust <- as.hclust(diana(d))

# convert to phylo class
p <- as.phylo(h.hclust)
```

```
library(soilDB)
data(loafercreek)
```

```
# slice all hz data into 1-cm slices, from 0-100cm
s.sliced <- slice(s, 0:100 ~ .)
plot(s.sliced)
```

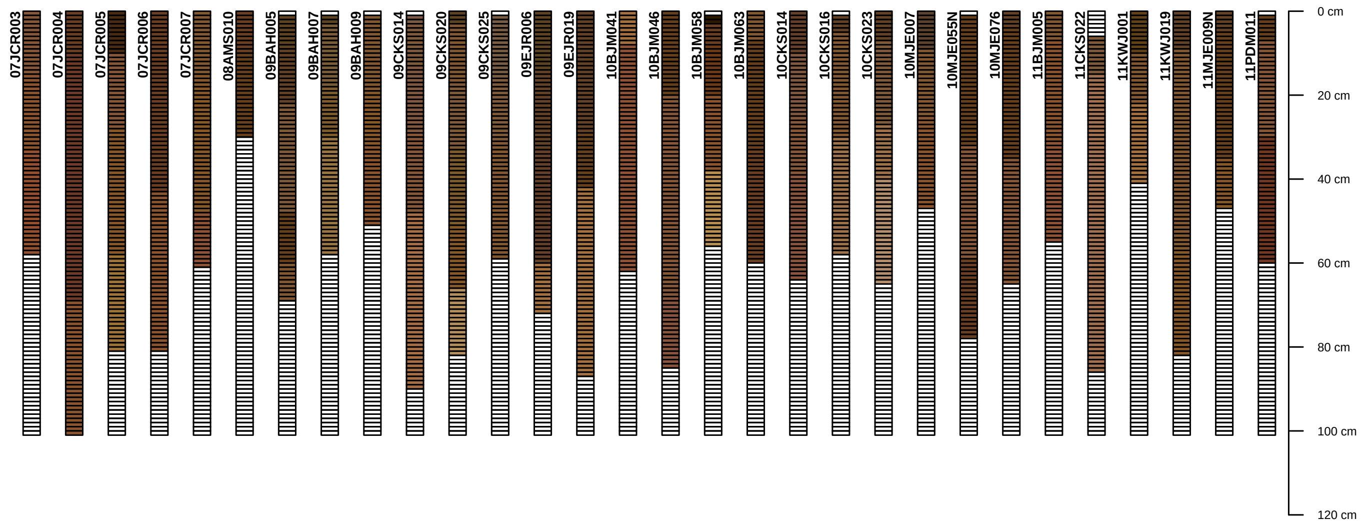

Figure 2: slicing, plotting

```
##
## aggregate major horizon types over 1cm intervals
##

# categorize major horizon types
hz.tab <- rev(sort(table(loafercreek$hzname)))
hz.tab[hz.tab > 5]

##
##    A Bt1 Bt2  Cr Bt3  Oi   R Crt
##   47  45  44  25  22  21  15  13


# add generalized hz name
loafercreek$hz <- rep("other", times = nrow(loafercreek))

# generalize horizons
loafercreek$hz[grep("O", loafercreek@horizons$hzname)] <- "O"
loafercreek$hz[grep("A", loafercreek@horizons$hzname)] <- "A"
loafercreek$hz[grep("Bt", loafercreek@horizons$hzname)] <- "Bt"
loafercreek$hz[grep("Cr", loafercreek@horizons$hzname)] <- "Cr"
loafercreek$hz[grep("R", loafercreek@horizons$hzname)] <- "R"

# convert generalized hz to factor
loafercreek$hz <- factor(loafercreek$hz)
loafercreek.hz.agg <- slab(loafercreek, fm = ~hz)

# wide->long format
loafercreek.hz.agg.long <- melt(loafercreek.hz.agg,
    id.var = c("top", "bottom", "contributing_fraction",
        "all.profiles", "variable"), variable_name = "horizon")
```

```
# extract and plot first 10 slices (0-10 cm)
plot(s.sliced[, 1:10], name = "hzname")
```

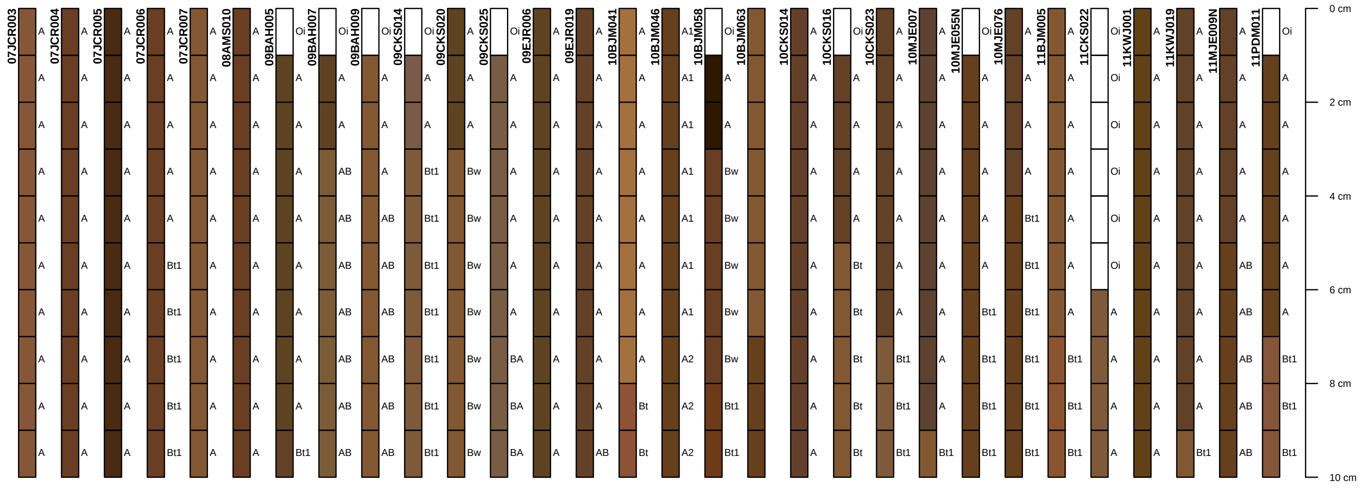

Figure 3: simple indexing

```
# plot horizon type proportions
p1 <- xyplot(top ~ value, groups = horizon, data = loafercreek.hz.agg.long,
    ylim = c(150, -5), type = c("S", "g"), horizontal = TRUE,
    subset = value > 0 & horizon != "other", asp = 2, ylab = "",
    xlab = "")
```

```
# extract and plot pedons 1-5, slices 25-50 (25-50 cm)
plot(s.sliced[1:5, 25:50])
```

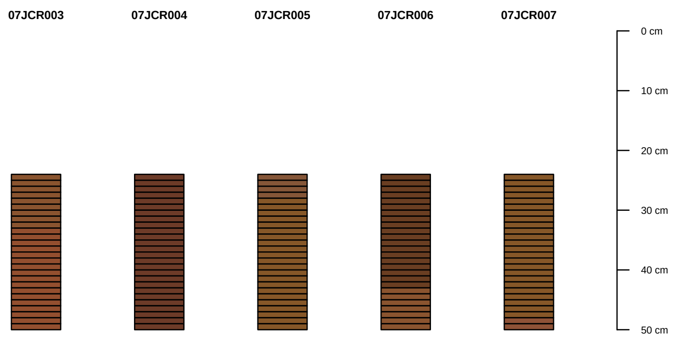

Figure 4: simple indexing + slice subsetting

```
# extract and plot pedons with paralithic
#   materials/contact, slices 50-75 (50-75 cm)
plot(s.sliced[grep("paralithic", s.sliced$diag_kind,
    ignore.case = TRUE), 50:75])
```

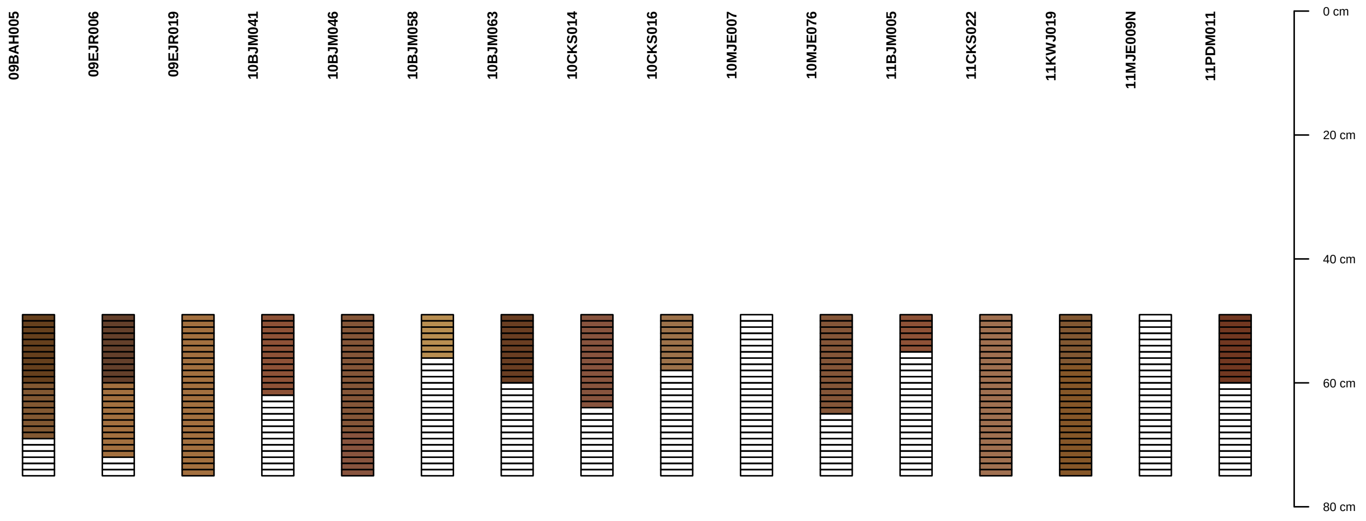

Figure 5: indexing based on query

```
## applying a function by profile ##

f <- function(x) {
    h <- horizons(x)
    max.clay <- which.max(h$clay)
    with(h[max.clay, ], (hzdept + hzdepb)/2)
}
max.clay.depth <- profileApply(s, f)

# plot and annotate with max clay depths
plot(s, name = "hzname")
x.pos <- 1:length(s)
points(x.pos, max.clay.depth, col = "white", pch = 21,
    cex = 0.75, bg = "black")
lines(x.pos, max.clay.depth, col = "black", lty = 2)
```

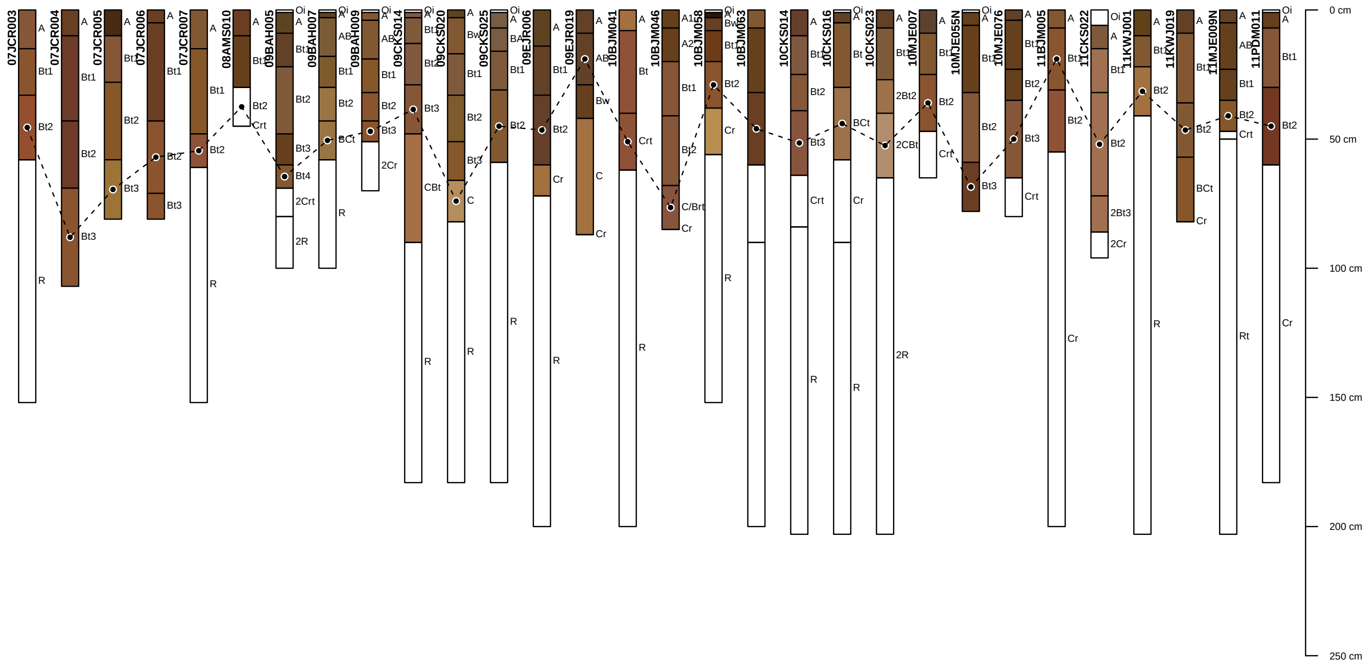

Figure 6: annotating profile plot with additional graphics

```
# all profiles: clay, sand, total RF
a <- slab(s, ~clay + sand + total_frags_pct)
a$mid <- with(a, (top + bottom)/2)

# plot median +/- IQR of clay, sand, total RF
xyplot(mid ~ p.q50 | variable, upper = a$p.q75,
    lower = a$p.q25, id = a$major_bedrock_kind, data = a,
    ylim = c(180, -5), ylab = "Depth (cm)", xlab = c("% Clay",
        "% Sand", "% Total RF"), strip = strip.custom(bg = "yellow"),
    par.settings = list(superpose.line = list(col = 1)),
    as.table = TRUE, panel = panel.depth_function, prepanel = prepanel.depth_function,
    scales = list(y = list(tick.number = 7, alternating = 3),
        x = list(alternating = 1)), layout = c(3, 1), cf = a$contributing_fraction)
```

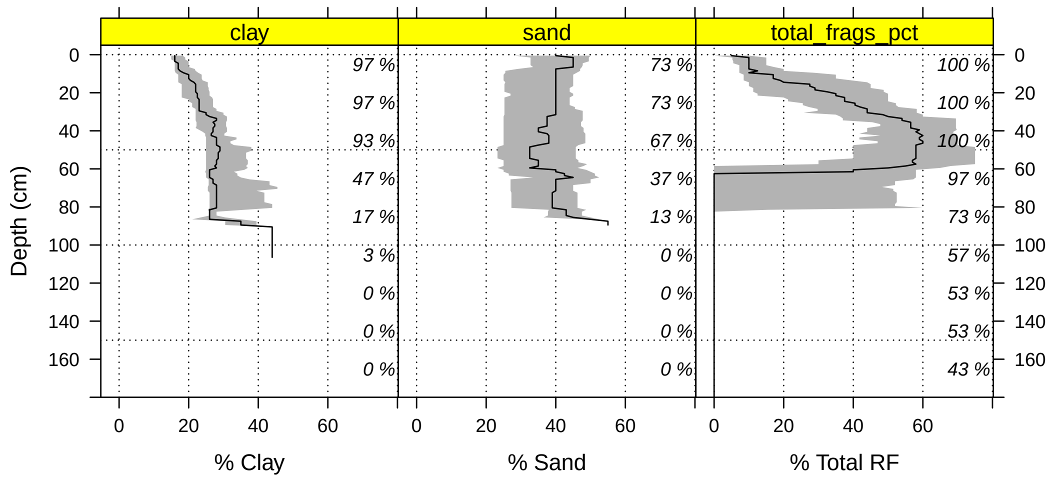

Figure 7: profile aggregation

```
# plot dendrogram + profiles
par(mar = c(0, 0, 0, 1))
p.plot <- plot(p, cex = 0.8, label.offset = -3,
    direction = "up", y.lim = c(85, 0), x.lim = c(1, length(s) +
        1), show.tip.label = FALSE)

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

# vector of indices for plotting soil profiles below
#   leaves of dendrogram
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
sf <- 0.15
yoff <- max(lastPP$yy) + 2.2

# add profiles below leaves of the dendrogram
plot(s, name = "hzname", color = "soil_color",
    plot.order = new_order, cex.id = 0.5, max.depth = 150,
    n.depth.ticks = 6, scaling.factor = sf, width = 0.1,
    cex.names = 0.5, cex.depth.axis = 0.75, y.offset = yoff,
    add = TRUE, id.style = "side")
```

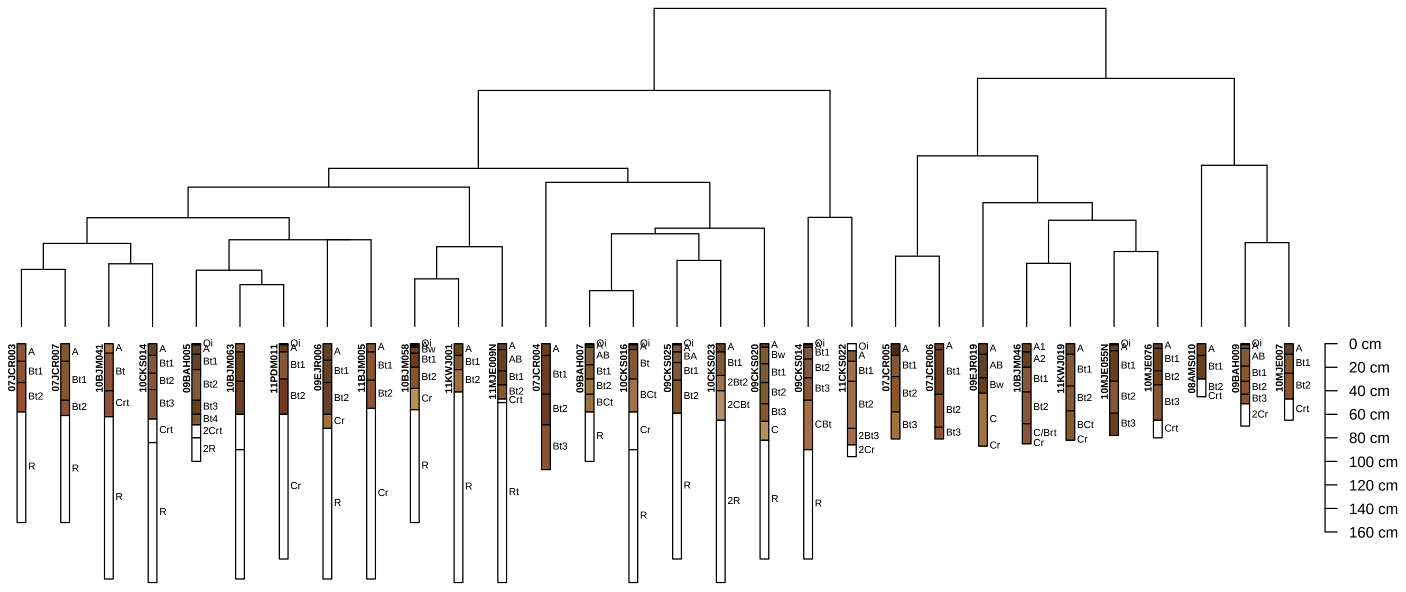

Figure 8: Dendrogram + profiles.

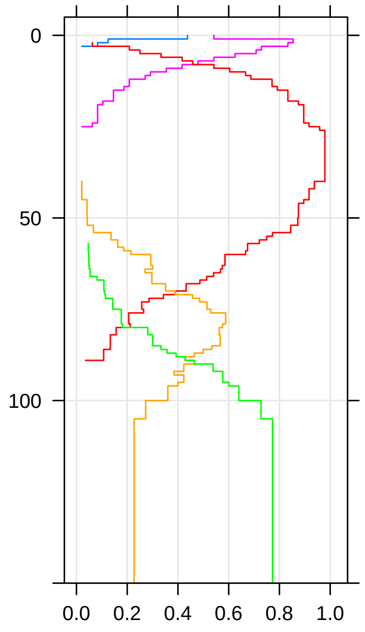

Figure 9: horizon proportions

```r
f.andic <- f[which(f$andic.soil.properties), ]

# extract and subset only andic soil properties
d <- diagnostic_hz(f.andic)
d <- d[d$diag_kind == "andic soil properties",
    c("pedon_id", "featdept", "featdepb")]


# join these data with existing site data
site(f.andic) <- d

# check: plot profiles
plot(f.andic, name = "hzname")
# add lines along top/bottom of 'andic soil properties'
#   diagnostic data
lines(1:length(f.andic), f.andic$featdept, lty = 2,
    lwd = 2)
lines(1:length(f.andic), f.andic$featdepb, lty = 2,
    lwd = 2)

# subset horizons, by profile, according to 'andic soil
#   properties' depths
# this is tricky, as it involves taking apart an SPC
# convert from SPC -> data.frame:
# 1 row / horizon, with site data joined with hz data
f.andic.df <- as(f.andic, "data.frame")
h.andic <- subset(f.andic.df, subset = hzdept >=
    featdept & hzdepb <= featdepb)

# check: OK
h.andic[, c("pedon_id", "hzname", "hzdept", "hzdepb",
    "featdept", "featdepb")]

# aggregate / stack / plot
# two collections of pedons
a.mvo <- slab(f[which(f$bedrock_kind == "Metavolcanics"),
    ], ~clay + sand + total_frags_pct)  # mvo data
a.slate <- slab(f[which(f$bedrock_kind == "Slate"),
    ], ~clay + sand + total_frags_pct)  # mvo data
a.andic <- slab(f.andic, ~clay + sand + total_frags_pct)  # andic soils

# stack:
a <- make.groups(metavolcanic = a.mvo, slate = a.slate,
    andic.soils = a.andic)
a$mid <- with(a, (top + bottom)/2)

# plot median +/- IQR of clay, sand, total RF
xyplot(mid ~ p.q50 | variable, groups = which,
    upper = a$p.q75, lower = a$p.q25, id = a$major_bedrock_kind,
    data = a, ylim = c(180, -5), ylab = "Depth (cm)", xlab = c("% Clay",
        "% Sand", "% Total RF"), strip = strip.custom(bg = grey(0.8)),
    par.settings = list(superpose.line = list(col = c("orange",
        "RoyalBlue", "Darkgreen"), lwd = 2)), as.table = TRUE,
    panel = panel.depth_function, prepanel = prepanel.depth_function,
    scales = list(y = list(tick.number = 7, alternating = 3),
        x = list(relation = "free", alternating = 1)), layout = c(3,
    1), sync.colors = TRUE, alpha = 0.5, auto.key = list(columns = 3,
    lines = TRUE, points = FALSE))
```