

# Soil Database Interface

## The `soilDB` package

D.E. Beaudette and J.M. Skovlin

June 19, 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.

```
# 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

```
library(soilDB)
# 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)

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

## 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

```
# libraries: you may have to install these
library(soilDB)
library(Hmisc)
library(ape)
library(lattice)
library(cluster)
library(reshape)

# load example dataset
data(gopheridge)

## aggregates of horizon data, moved into @site

# soil depth using an in-line function, note that max(SPC) returns the
# bottom-most depth this may not be the same as 'soil depth' when Cr or R
# horizons are present
gopheridge$soil.depth <- profileApply(gopheridge, function(x) max(x))

# 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)
}

# get hz mid-point of hz with max clay content
f.max.clay.depth <- function(x) {
  h <- horizons(x)
```

```

    max.clay <- which.max(h$clay)
    with(h[max.clay, ], (hzdept + hzdepb)/2)
  }

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

# 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)
}

# conditional eval of top hz depth of diagnostic feature or horizon will
# return a vector of length(x), you can save to @site
f.diag.top <- 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)

  # return top depth
  return(d$featdept)
}

# compute wt.mean clay within PCS, save to @site
gopheridge$pcs.clay <- profileApply(gopheridge, f.pcs.prop, prop = "clay")

# compute wt.mean clay within argillic horizon, save to @site
gopheridge$argillic.clay <- profileApply(gopheridge, f.diag.wt.prop,
  d.hz = "argillic horizon", prop = "clay")

# compute argillic hz thickness, save to @site
gopheridge$argillic.thick <- profileApply(gopheridge, f.diag.thickness,
  d.hz = "argillic horizon")

# compute depth to paralithic contact, if present, save to @site
gopheridge$paralithic.contact.depth <- profileApply(gopheridge, f.diag.top,
  d.hz = "paralithic contact")

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

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

##          id clay sand total_frgs_pct
## 1    07JCR002   38   38           0
## 2    07JCR003   25   25           0
## 3    07JCR004    0  100           0

```

```
# order by presence of paralithic contact
new.order <- order(gopheridge$paralithic.contact)
plot(gopheridge, name = "hzname", plot.order = new.order)
# annotate paralithic contact with lines
x.pos <- 1:length(gopheridge)
lines(x.pos, gopheridge$paralithic.contact.depth[new.order], lty = 2,
      col = "black")
```

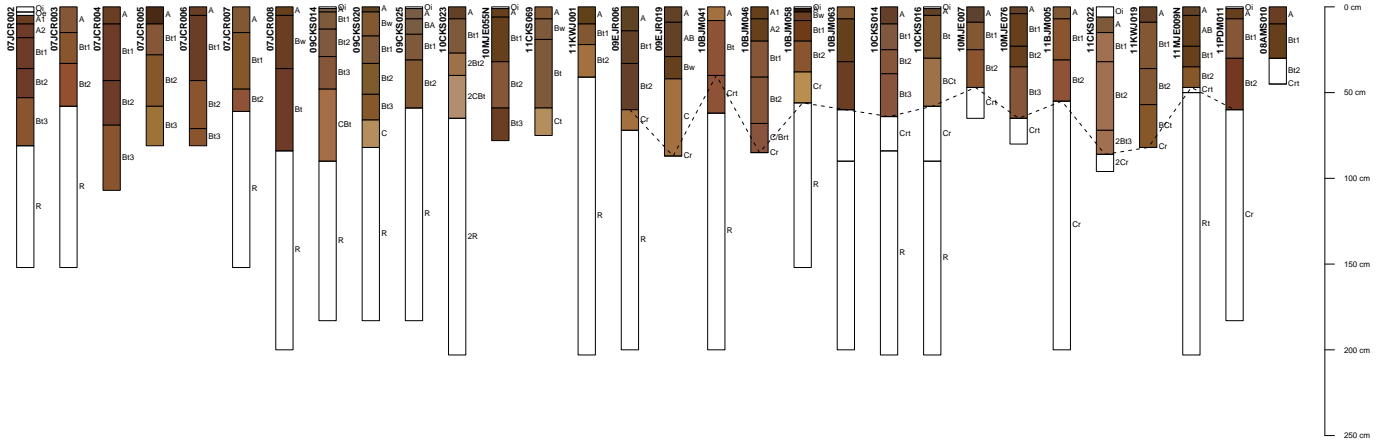


Figure 1: Basic plotting

```
## 6 07JCR007 25 50 0
## 7 07JCR008 25 25 0
## 8 08AMS010 25 100 0
## 9 09CKS014 29 29 0
## 10 09CKS020 14 14 0
## 11 09CKS025 33 33 0
## 12 09EJR006 20 20 0
## 13 09EJR019 20 20 0
## 14 10BJM041 25 25 0
## 15 10BJM046 17 17 0
## 16 10BJM058 43 43 0
## 17 10BJM063 40 40 0
## 18 10CKS014 33 33 0
## 19 10CKS016 50 50 0
## 20 10CKS023 20 20 0
## 21 10MJE007 25 25 0
## 22 10MJE055N 20 100 0
## 23 10MJE076 20 20 0
## 24 11BJM005 25 25 0
## 25 11CKS022 33 33 0
## 27 11KWJ001 25 25 0
## 28 11KWJ019 20 20 0
## 29 11MJE009N 50 100 0
## 30 11PDM011 40 40 0
```

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

# convert to phylo class
```

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

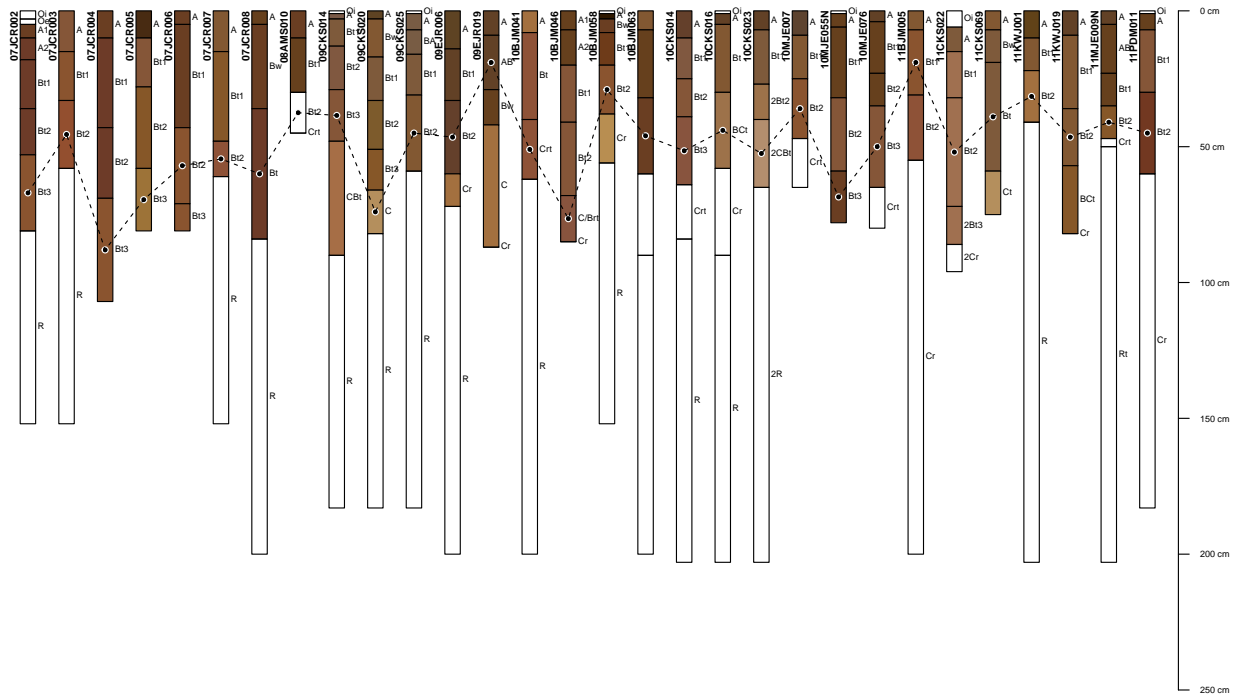


Figure 2: annotating profile plot with additional graphics

```
p <- as.phylo(h.hclust)
```

```
data(loafercreek)

# 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 BAt
##  52  50  48  28  23  22  16  14   6

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

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

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

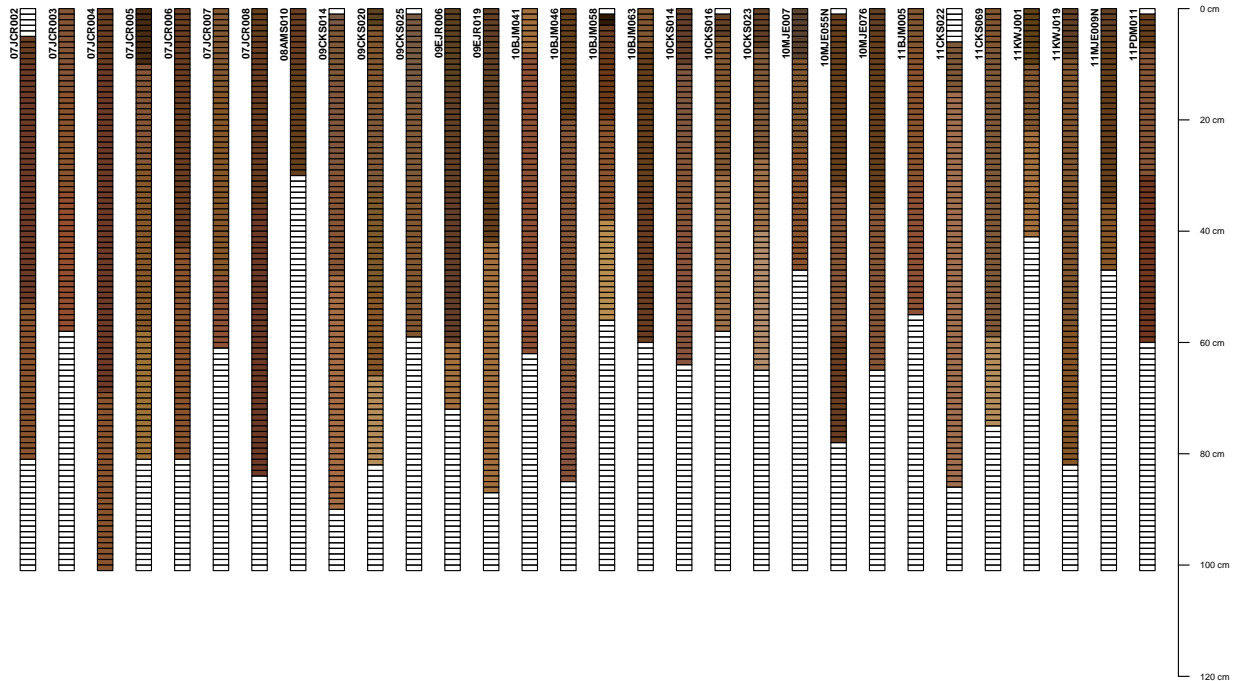


Figure 3: slicing, plotting

```
loafercreek$hz[grep("R", loafercreek$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")

# 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, asp = 1.5, ylab = "", xlab = "", auto.key = list(columns = 5, lines = TRUE,
    points = FALSE))
```

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

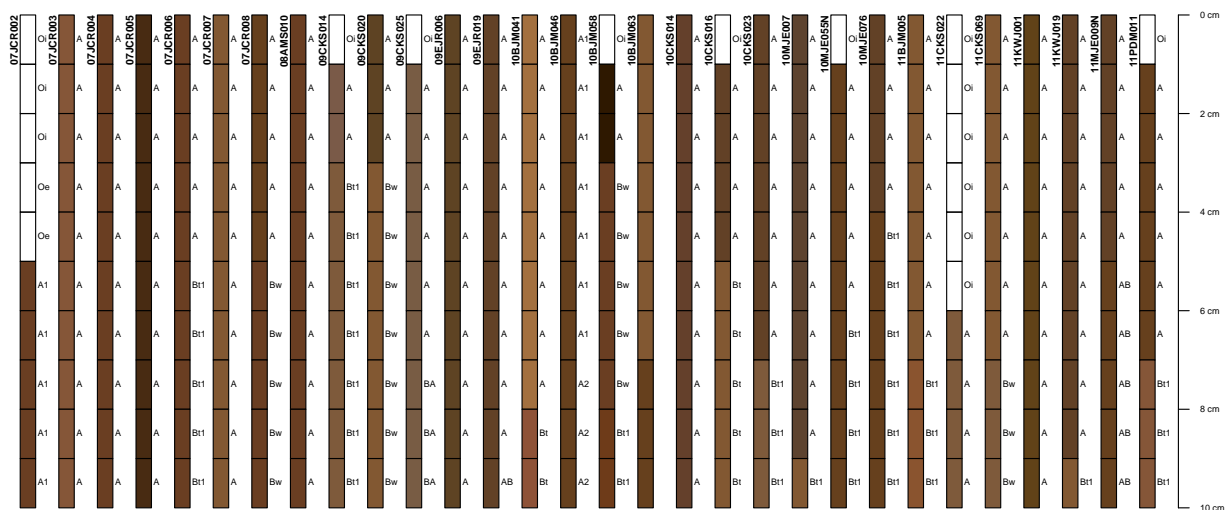


Figure 4: simple indexing



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

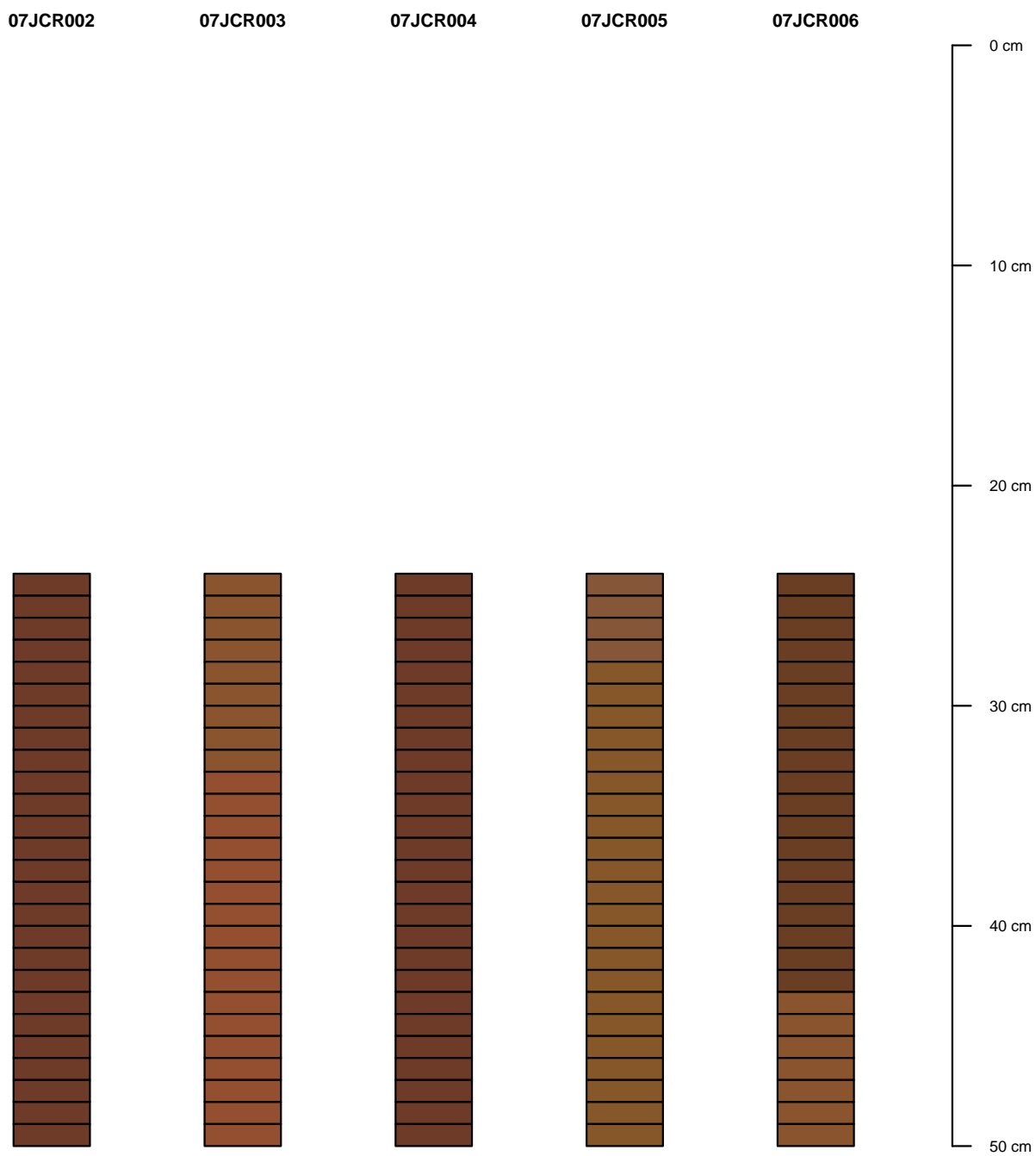


Figure 5: simple indexing + slice subsetting

```
# extract and plot pedons with paralithic materials/contact, slices 50-75
# (50-75 cm)
plot(gopheridge.sliced[which(gopheridge$paralithic.contact), 50:75])
```

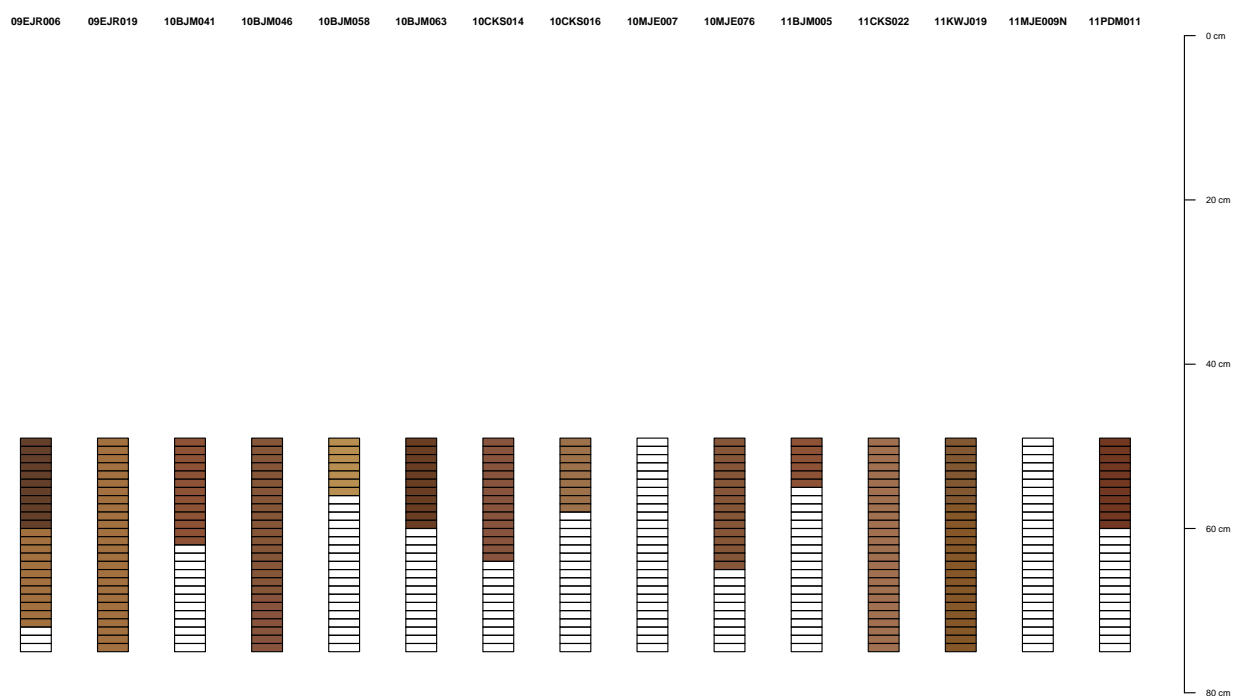


Figure 6: indexing based on query

```

# all profiles: clay, sand, total RF
a <- slab(gopheridge, ~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)

```

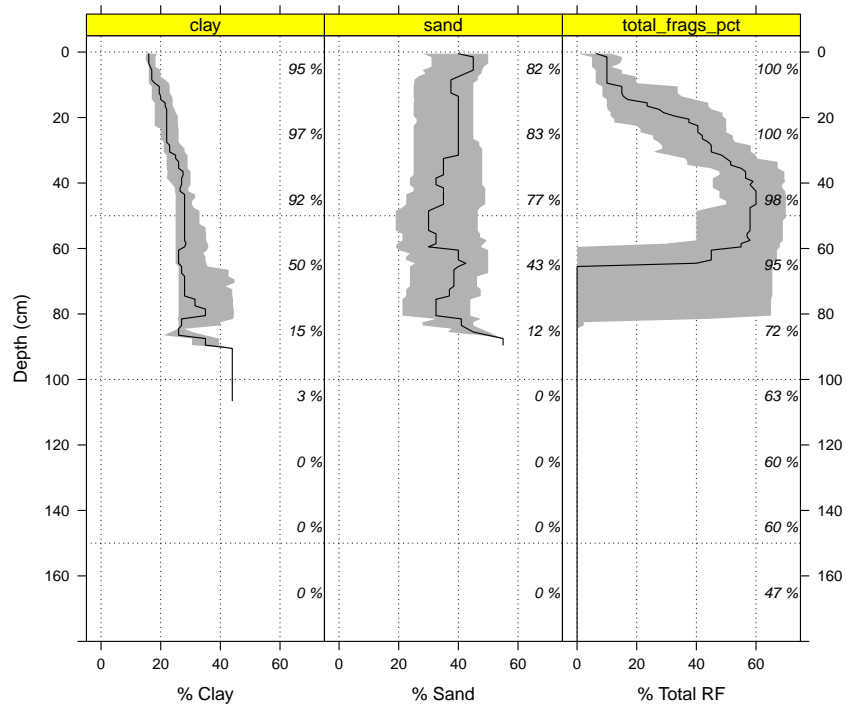


Figure 7: profile aggregation



```
print(p1)
```

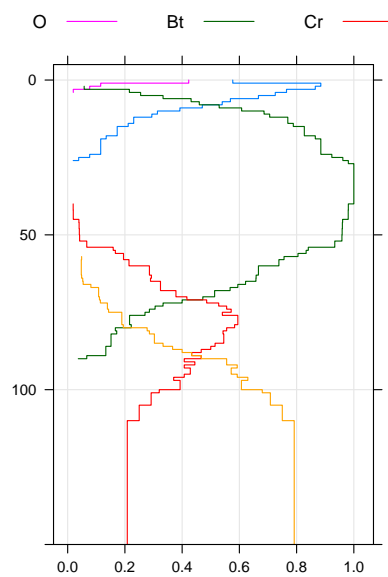


Figure 9: horizon proportions

```

## compute dissimilarity 2 ways: 1) with hz data, 2) with hz+site data

data(loafercreek)

# just hz-level variables: need at least 2
d.hz <- profile_compare(loafercreek, vars = c("clay", "total_frgs_pct"),
  k = 0, max_d = 100, rescale.result = TRUE)

# hz and site level variables: need at least 2 hz-level and 2 site-level
# character-class variables should be manually converted to factors first
# !!
d.site.hz <- profile_compare(loafercreek, vars = c("clay", "total_frgs_pct",
  "paralithic_contact", "slope"), k = 0, max_d = 100, rescale.result = TRUE)

# note: both dissimilarity matrices have been rescaled to [0,1]
range(d.hz)
range(d.site.hz)

# cluster via divisive hierarchical algorithm convert to 'phylo' class
phylo.hz <- as.phylo(as.hclust(diana(d.hz)))
phylo.site.hz <- as.phylo(as.hclust(diana(d.site.hz)))

# generate vector of labels for site-level clustering criteria paralithic
# contact = 1, missing = 2
paralithic.contact.code <- as.integer(loafercreek$paralithic.contact) +
  1

# setup some geometry since our dissimilarity matrices are scaled to [0,1]
# hard-code some limits
arrow.left <- 0.1
arrow.right <- 0.9
arrow.length <- 0.1

# setup plot layout
lo <- layout(matrix(c(1, 3, 2), ncol = 3), widths = c(1, 1, 1)) # layout.show(lo)

# plot hz-level dissimilarities on left-hand side
plot(phylo.hz, cex = 0.75, font = 1, no.margin = TRUE, direction = "rightwards",
  label.offset = 0.015)
mtext("Horizon Data", side = 3, line = -1.5, font = 2, cex = 0.75)

# save results of phylo environment
p.left <- get("last_plot.phylo", envir = .PlotPhyloEnv)
# annotate pedons with paralithic contact
tiplabels(pch = 15, col = c("orange", NA)[paralithic.contact.code])

# plot site+hz-level dissimilarities on right-hand side
plot(phylo.site.hz, cex = 0.75, font = 1, direction = "leftwards",
  no.margin = TRUE, label.offset = 0.015)
mtext("Horizon + Site Data", side = 3, line = -1.5, font = 2, cex = 0.75)

# save results of phylo environment
p.right <- get("last_plot.phylo", envir = .PlotPhyloEnv)
# annotate pedons with paralithic contact
tiplabels(pch = 15, col = c("orange", NA)[paralithic.contact.code])

```

```

# get ordering of pedon IDs on each side left-hand side
left.new_order <- sapply(1:p.left$Ntip, function(i) which(as.integer(p.left$yy[1:p.left$Ntip]) ==
  i))
# right-hand side
right.new_order <- sapply(1:p.right$Ntip, function(i) which(as.integer(p.right$yy[1:p.right$Ntip]) ==
  i))

# re-order right-hand side nodes so that they match the ordering of
# left-side nodes
left.right.conversion.order <- right.new_order[match(left.new_order,
  right.new_order)]

# setup new plot region
par(mar = c(0, 0, 0, 0))
plot(0, 0, type = "n", xlim = c(0, 1), ylim = range(p.left$yy), axes = FALSE,
  xpd = FALSE)

# plot connecting segments
segments(x0 = arrow.left, y0 = p.left$yy[left.new_order], x1 = arrow.right,
  y1 = p.right$yy[left.right.conversion.order], col = "RoyalBlue")

# plot helper arrows
arrows(x0 = arrow.left, y0 = p.left$yy[left.new_order], x1 = arrow.left -
  arrow.length, y1 = p.left$yy[left.new_order], col = "RoyalBlue", code = 2,
  length = 0.05)
arrows(x0 = arrow.right, y0 = p.right$yy[right.new_order], x1 = arrow.right +
  arrow.length, y1 = p.right$yy[right.new_order], col = "RoyalBlue", code = 2,
  length = 0.05)

# legend: top-center
legend("top", legend = c("Paralithic Contact"), pch = 15, col = "orange",
  bty = "n")

```

```

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$bedrock_kind == "Metavolcanics"), ], ~clay +
sand + total_frgs_pct) # mvo data
a.slate <- slab(f[which(f$bedrock_kind == "Slate"), ], ~clay + sand +
total_frgs_pct) # mvo data
a.andic <- slab(f.andic, ~clay + sand + total_frgs_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))

```