Package 'simplex'

November 3, 2019

Title Data Reduction Software for Secondary Ion Mass Spectrometry

Version 0.1

Date 2019-10-28

Description Processes Secondary Ion Mass Spectrometery (SIMS) data within the confines of the simplex, i.e. the data space of compositions. Accommodates input files for both Cameca and SHRIMP instruments. Models the data using a combination of multinomial and logistic normal statistics. Keeps track of inter-sample error correlations caused by using a common standard for multiple samples. Includes applications for U-Pb geochronology and stable isotope geochemistry.

Author Pieter Vermeesch [aut, cre]

Maintainer Pieter Vermeesch <p.vermeesch@ucl.ac.uk>

Depends R (>= 3.0.0)

Imports MASS, graphics, IsoplotR

License GPL-3

URL https://github.com/pvermees/simplex

LazyData true

RoxygenNote 6.1.1

Encoding UTF-8

R topics documented:

`simplex-package` *Data Reduction Software for Secondary Ion Mass Spectrometry*

Description

Processes Secondary Ion Mass Spectrometery (SIMS) data within the confines of the simplex, i.e. the data space of compositions. Accommodates input files for both Cameca and SHRIMP instruments. Models the data using a combination of multinomial and logistic normal statistics. Keeps track of inter-sample error correlations caused by using a common standard for multiple samples. Includes applications for U-Pb geochronology and stable isotope geochemistry.

Details

The DESCRIPTION file: This package was not yet installed at build time.

Index: This package was not yet installed at build time.
~~ An overview of how to use the package, including the most important ~~ ~~ functions ~~

Author(s)

Pieter Vermeesch [aut, cre]

Maintainer: Pieter Vermeesch <p.vermeesch@ucl.ac.uk>

References

~~ Literature or other references for background information ~~

See Also

~~ Optional links to other man pages, e.g. ~~ ~~ <pkg> ~~

Examples

```
~~ simple examples of the most important functions ~~
```

---

ag2b

Usage

```
ag2b(ag, samp, c64 = NULL)
```

Arguments

ag

samp

c64

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (ag, samp, c64 = NULL)
{
    simplex <- c("204Pb", "206Pb", "207Pb", "238U", "238U16O2")
    tt <- hours(samp$time[, simplex])
    cc <- samp$counts[, simplex]
    dt <- samp$dwelltime[simplex]
    fn6 <- cc[, "206Pb"] ~ 1 + offset(ag["g"] * tt[, "206Pb"])
    fn7 <- cc[, "207Pb"] ~ 1 + offset(ag["g"] * tt[, "207Pb"])
    fnU <- cc[, "238U"] ~ 1 + offset(ag["g"] * tt[, "238U"])
    fnUO <- cc[, "238U16O2"] ~ 1 + offset(ag["g"] * tt[, "238U16O2"])
    b6 <- glm(fn6, family = poisson(link = "log"))$coef
    b7 <- glm(fn7, family = poisson(link = "log"))$coef
    bU <- glm(fnU, family = poisson(link = "log"))$coef
    bUO <- glm(fnUO, family = poisson(link = "log"))$coef
    if (is.null(c64)) {
        b4 <- b6 - log(exp(ag["a4"]) + 1)
    }
    else {
        b4 <- b6 - log(c64 * dt["206Pb"]/dt["204Pb"]) - log(exp(ag["a4"]) +
            1)
    }
    out <- c(b4, b6, b7, bU, bUO)
```

```
  names(out) <- simplex
  out
}
```

---

ages

Usage

```
ages(logPbU)
```

Arguments

```
logPbU
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (logPbU)
{
    out <- list()
    Pb6U8 <- exp(logPbU$x)
    JR <- diag(Pb6U8)
    E <- JR %*% logPbU$cov %*% t(JR)
    l38 <- IsoplotR::settings("lambda", "U238")[1]
    out$x <- log(1 + Pb6U8)/l38
    Jt <- diag(1/(l38 * (1 + Pb6U8)))
    out$cov <- Jt %*% E %*% t(Jt)
    labels <- names(logPbU$x)
    names(out$x) <- labels
    rownames(out$cov) <- labels
    colnames(out$cov) <- labels
    out
  }
```

---

avg_Lm

Usage

```
avg_Lm(Lm)
```

Arguments

```
Lm
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (Lm)
{
    np <- length(Lm[[1]]$x)
    init <- rep(0, np)
    nt <- length(Lm)
    for (i in 1:nt) {
        init <- init + Lm[[i]]$x/nt
    }
    lower <- init - rep(1, np)
    upper <- init + rep(1, np)
    fit <- optim(init, misfit_avg_Lm, Lm = Lm, lower = lower,
        upper = upper, method = "L-BFGS-B", hessian = TRUE)
    out <- list()
    out$x <- fit$par
    out$cov <- solve(fit$hessian)
    labels <- c("L6m", "L7m", "LUm")
    names(out$x) <- labels
    rownames(out$cov) <- labels
    colnames(out$cov) <- labels
    out
  }
```

---

b2p

Usage

```
b2p(b, ag, tt)
```

Arguments

b
ag
tt

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (b, ag, tt)
```

```
{
    simplex <- c("204Pb", "206Pb", "207Pb", "238U", "238U16O2")
    nr <- nrow(tt)
    bm <- matrix(rep(b, nr), nrow = nr, byrow = TRUE)
    colnames(bm) <- simplex
    ebgt <- exp(bm + ag["g"] * tt)
    out <- ebgt/sum(ebgt)
    colnames(out) <- simplex
    out
  }
```

---

bias_correction

Usage

```
bias_correction(samp, aLm, ag, c64 = NULL)
```

Arguments

samp

aLm

ag

c64

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (samp, aLm, ag, c64 = NULL)
{
    simplex <- c("204Pb", "206Pb", "207Pb", "238U", "238U16O2")
    tt <- hours(samp$time[, simplex])
    dt6 <- mean(tt[, "206Pb"] - tt[, "238U"])
    dt7 <- mean(tt[, "207Pb"] - tt[, "206Pb"])
    dtU <- mean(tt[, "238U16O2"] - tt[, "238U"])
    L6c <- aLm$x["L6m"] - ag$x["g"] * dt6
    L7c <- aLm$x["L7m"] - ag$x["g"] * dt7
    LUc <- aLm$x["LUm"] - ag$x["g"] * dtU
    if (is.null(c64)) {
        L4c <- log(exp(ag$x["a4"]) + 1)
    }
    else {
        d4 <- samp$dwelltime["204Pb"]
        d6 <- samp$dwelltime["206Pb"]
        L4c <- log(exp(ag$x["a4"]) + 1) + log(c64 * d6/d4)
```

```
    }
    out <- list()
    out$x <- c(L4c, L6c, L7c, LUc)
    labels <- c("L4c", "L6c", "L7c", "LUc")
    names(out$x) <- labels
    E <- matrix(0, 5, 5)
    E[1:3, 1:3] <- aLm$cov
    E[4:5, 4:5] <- ag$cov
    J <- matrix(0, 4, 5)
    rownames(J) <- labels
    colnames(J) <- c("L6m", "L7m", "LUm", "a4", "g")
    J["L6c", "L6m"] <- 1
    J["L7c", "L7m"] <- 1
    J["LUc", "LUm"] <- 1
    J["L4c", "a4"] <- exp(ag$x["a4"])/(exp(ag$x["a4"]) + 1)
    J["L6c", "g"] <- -dt6
    J["L7c", "g"] <- -dt7
    J["LUc", "g"] <- -dtU
    out$cov <- J %*% E %*% t(J)
    out
  }
```

---

calibrate

Usage

```
calibrate(lr, fit, dat, PbUstand = NULL, tst = c(337.13, 0.18))
```

Arguments

lr
fit
dat
PbUstand
tst

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (lr, fit, dat, PbUstand = NULL, tst = c(337.13, 0.18))
{
    XY <- lr2XY(lr = lr, dat = dat)
    xy <- flatXYtable(XY)
    ns <- nrow(xy)
```

```
  if (is.null(PbUstand))
      Pb6U8stand <- IsoplotR:::age_to_Pb206U238_ratio(tt = tst[1],
          st = tst[2])
  Ytstand <- log(Pb6U8stand[1])
  out <- list()
  out$x <- xy[, "Y"] + Ytstand - fit$AB["A"] - fit$AB["B"] *
      xy[, "X"]
  sYtstand <- Pb6U8stand[2]/Pb6U8stand[1]
  E <- matrix(0, 2 * ns + 3, 2 * ns + 3)
  i1 <- 1:ns
  i2 <- (ns + 1):(2 * ns)
  E[i1, i1] <- diag(xy[, "sX"]^2)
  E[i2, i2] <- diag(xy[, "sY"]^2)
  E[i1, i2] <- diag(xy[, "rXY"] * xy[, "sX"] * xy[, "sY"])
  E[i2, i1] <- E[i1, i2]
  E[(2 * ns + 1), (2 * ns + 1)] <- sYtstand^2
  E[(2 * ns) + (2:3), (2 * ns) + (2:3)] <- fit$cov
  J <- matrix(0, nrow = ns, ncol = 2 * ns + 3)
  rownames(J) <- rownames(xy)
  J[i1, i1] <- diag(-fit$AB["B"], ns, ns)
  J[i1, i2] <- diag(1, ns, ns)
  J[, 2 * ns + 1] <- 1
  J[, 2 * ns + 2] <- -1
  J[, 2 * ns + 3] <- -xy[, "X"]
  out$cov <- J %*% E %*% t(J)
  out
}
```

---

calibration

Usage

```
calibration(lr, dat, plot = TRUE, disp = TRUE, omit = NULL, ...)
```

Arguments

lr	
dat	
plot	
disp	
omit	
...	

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (lr, dat, plot = TRUE, disp = TRUE, omit = NULL, ...)
{
    XY <- lr2XY(lr = lr, dat = dat)
    fit <- yorkfit(XY, omit = omit)
    out <- list()
    out$AB <- fit$par
    out$df <- length(dat) - 2
    out$mswd <- 2 * fit$value/out$df
    if (disp)
        d <- out$mswd
    else d <- 1
    out$cov <- d * solve(fit$hessian)
    labels <- c("A", "B")
    names(out$AB) <- labels
    rownames(out$cov) <- labels
    colnames(out$cov) <- labels
    if (plot)
        calplot(XY, fit = out, omit = omit)
    out
  }
```

---

calplot

Usage

```
calplot(XY, fit, alpha = 0.05, sigdig = 2, omit = NULL, ...)
```

Arguments

XY

fit

alpha

sigdig

omit

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (XY, fit, alpha = 0.05, sigdig = 2, omit = NULL, ...)
{
    xy <- flatXYtable(XY)
    isofit <- list(model = 1)
    isofit$fact <- stats::qt(1 - alpha/2, fit$df)
    isofit$n <- nrow(xy)
    isofit$mswd <- fit$mswd
    isofit$p.value <- as.numeric(1 - stats::pchisq(fit$mswd/fit$df,
        fit$df))
    A <- fit$AB["A"]
    B <- fit$AB["B"]
    sA <- sqrt(fit$cov["A", "A"])
    sB <- sqrt(fit$cov["B", "B"])
    isofit$a <- c(A, sA, isofit$fact * sA, isofit$mswd * isofit$fact *
        sA)
    isofit$b <- c(B, sB, isofit$fact * sB, isofit$mswd * isofit$fact *
        sB)
    isofit$cov.ab <- fit$cov["A", "B"]
    xlab <- quote(""^238 * "U"^16 * "O"[2] * "/"^238 * "U")
    ylab <- quote(""^206 * "Pb*/"^238 * "U")
    scatterplot(xy, xlab = xlab, ylab = ylab, fit = isofit, omit = omit,
        ...)
    graphics::title(isochrontitle(isofit, sigdig = sigdig), xlab = xlab,
        ylab = ylab)
  }
```

---

calplot_raw

Usage

```
calplot_raw(dat, i = NULL, c64 = NULL, ...)
```

Arguments

dat	
i	
c64	
...	

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (dat, i = NULL, c64 = NULL, ...)
{
    vars <- reshuffle(dat)
    nt <- nrow(dat[[1]]$time)
    ns <- length(dat)
    if (is.null(i))
        ii <- 1:ns
    else ii <- i
    matplot(vars$X[, ii], vars$Y[, ii], type = "l", ...)
    text(vars$X[, ii], vars$Y[, ii], labels = 1:nt)
  }
```

---

flatXYtable

Usage

```
flatXYtable(XY)
```

Arguments

```
XY
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (XY)
{
    snames <- names(XY)
    ns <- length(snames)
    xy <- matrix(0, ns, 5)
    colnames(xy) <- c("X", "sX", "Y", "sY", "rXY")
    for (i in 1:ns) {
        xy[i, "X"] <- XY[[i]]$x["X"]
        xy[i, "Y"] <- XY[[i]]$x["Y"]
        xy[i, "sX"] <- sqrt(XY[[i]]$cov["X", "X"])
        xy[i, "sY"] <- sqrt(XY[[i]]$cov["Y", "Y"])
        xy[i, "rXY"] <- XY[[i]]$cov["X", "Y"]/(xy[i, "sX"] *
            xy[i, "sY"])
    }
```

```
  xy
}
```

---

get_ag

Usage

```
get_ag(samp, c64 = NULL)
```

Arguments

samp

c64

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (samp, c64 = NULL)
{
    init <- init_ag(samp, c64 = c64)
    np <- length(init)
    lower <- init - rep(1, np)
    upper <- init + rep(1, np)
    fit <- optim(init, misfit_ag, method = "L-BFGS-B", lower = lower,
        upper = upper, samp = samp, c64 = c64, control = list(fnscale = -1),
        hessian = TRUE)
    out <- list()
    out$x <- fit$par
    out$cov <- solve(-fit$hessian)
    out
  }
```

---

hours

Usage

```
hours(tt)
```

Arguments

tt

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (tt)
{
    tt/3600
  }
```

---

init_ag

Usage

```
init_ag(samp, c64 = NULL)
```

Arguments

samp

c64

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (samp, c64 = NULL)
{
    t4 <- hours(samp$time[, "204Pb"])
    t6 <- hours(samp$time[, "206Pb"])
    c4 <- samp$counts[, "204Pb"]
    c6 <- samp$counts[, "206Pb"]
    fit6 <- glm(c6 ~ t6, family = poisson(link = "log"))
    b6 <- fit6$coef[1]
    g <- fit6$coef[2]
    b4 <- glm(c4 ~ 1 + offset(g * t4), family = poisson(link = "log"))$coef
    if (is.null(c64)) {
        a4 <- log(exp(b6 - b4) - 1)
    }
    else {
        d4 <- samp$dwelltime["204Pb"]
        d6 <- samp$dwelltime["206Pb"]
        a4 <- log(exp(b6 - b4) * (d4/d6)/c64 - 1)
    }
    out <- c(a4, g)
```

```
  names(out) <- c("a4", "g")
  out
}
```

---

logratios

Usage

```
logratios(dat, c64 = NULL)
```

Arguments

dat

c64

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (dat, c64 = NULL)
{
    snames <- names(dat)
    ns <- length(snames)
    out <- list()
    for (sname in snames) {
        print(sname)
        samp <- dat[[sname]]
        Lm <- raw_count_ratios(samp = samp)
        aLm <- avg_Lm(Lm)
        ag <- get_ag(samp = samp, c64 = c64)
        out[[sname]] <- bias_correction(samp = samp, aLm = aLm,
            ag = ag)
    }
    out
}
```

lr2XY

Usage

```
lr2XY(lr, dat, c64 = NULL)
```

Arguments

lr	
dat	
c64	

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (lr, dat, c64 = NULL)
{
    snames <- names(dat)
    ns <- length(snames)
    out <- list()
    labels <- c("X", "Y")
    calions <- c("L4c", "L6c", "LUc")
    J <- matrix(0, 2, 3)
    rownames(J) <- labels
    colnames(J) <- calions
    J["X", "LUc"] <- 1
    J["Y", "L6c"] <- 1
    for (sname in snames) {
        d4 <- dat[[sname]]$dwelltime["204Pb"]
        d6 <- dat[[sname]]$dwelltime["206Pb"]
        dU <- dat[[sname]]$dwelltime["238U"]
        dUO <- dat[[sname]]$dwelltime["238U16O2"]
        L4c <- lr[[sname]]$x["L4c"]
        L6c <- lr[[sname]]$x["L6c"]
        LUc <- lr[[sname]]$x["LUc"]
        X <- LUc + log(dU/dUO)
        if (is.null(c64)) {
            Y <- L6c + log(dU/d6)
        }
        else {
            Y <- log(dU) + log(dU/d6) + log(1 - (c64 * d6)/(d4 *
                exp(L4c)))
            J["Y", "L4c"] <- c64 * d6/(c64 * d6 + exp(L4c) *
                d4)
```

```
        }
        out[[sname]] <- list()
        out[[sname]]$x <- c(X, Y)
        names(out[[sname]]$x) <- labels
        out[[sname]]$cov <- J %*% lr[[sname]]$cov[calions, calions] %*%
            t(J)
        out[[sname]]$omega <- solve(out[[sname]]$cov)
    }
    out
  }
```

---

misfit_ag

Usage

```
misfit_ag(ag, samp, c64 = NULL)
```

Arguments

ag

samp

c64

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (ag, samp, c64 = NULL)
{
    simplex <- c("204Pb", "206Pb", "207Pb", "238U", "238U16O2")
    b <- ag2b(ag, samp = samp, c64 = c64)
    p <- b2p(b, ag["g"], tt = hours(samp$time[, simplex]))
    counts <- samp$counts[, simplex]
    dmultinom(counts, prob = p, log = TRUE)
  }
```

misfit_avg_Lm

Usage

```
misfit_avg_Lm(aLm, Lm)
```

Arguments

aLm

Lm

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (aLm, Lm)
{
    nt <- length(Lm)
    out <- 0
    for (i in 1:nt) {
        D <- Lm[[i]]$x - aLm
        out <- out - D %*% Lm[[i]]$H %*% D
    }
    out/2
  }
```

misfit_york

Usage

```
misfit_york(AB, XY = XY)
```

Arguments

AB

XY

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (AB, XY = XY)
{
    A <- AB[1]
    B <- AB[2]
    SS <- 0
    snames <- names(XY)
    D <- matrix(0, 1, 2)
    for (sname in snames) {
        X <- XY[[sname]]$x["X"]
        Y <- XY[[sname]]$x["Y"]
        O <- XY[[sname]]$omega
        CC <- Y - A - B * X
        C1 <- O[1, 1] + O[1, 2] * B + O[2, 1] * B + O[2, 2] *
            B^2
        C2 <- 2 * (O[1, 2] + O[2, 2] * B) * CC
        C3 <- O[2, 2] * CC^2
        K <- -C2/(2 * C1)
        SS <- SS + C1 * K^2 + C2 * K + C3
    }
    SS/2
  }
```

---

plot_timeresolved

Usage

```
plot_timeresolved(samp, fit = FALSE, c64 = NULL)
```

Arguments

samp	
fit	
c64	

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (samp, fit = FALSE, c64 = NULL)
```

```
{
    ions <- names(samp$dwelltime)
    np <- length(ions)
    nr <- ceiling(sqrt(np))
    nc <- ceiling(np/nr)
    par(mfrow = c(nr, nc), mai = c(0.4, 0.4, 0.1, 0.1))
    simplex <- c("204Pb", "206Pb", "207Pb", "238U", "238U16O2")
    if (fit) {
        X <- samp$time[, simplex]
        Y <- predict_cps(samp, c64 = c64)[, simplex]
    }
    for (ion in ions) {
        plot(samp$time[, ion], samp$cps[, ion], type = "p", xlab = "",
            ylab = "")
        if (fit & ion %in% simplex) {
            lines(X[, ion], Y[, ion])
        }
        mtext(side = 1, text = "t", line = 2)
        mtext(side = 2, text = ion, line = 2)
    }
  }
```

---

predict_cps

Usage

```
predict_cps(samp, c64 = NULL)
```

Arguments

samp

c64

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (samp, c64 = NULL)
{
    simplex <- c("204Pb", "206Pb", "207Pb", "238U", "238U16O2")
    Lm <- raw_count_ratios(samp = samp)
    aLm <- avg_Lm(Lm)
    ag <- get_ag(samp = samp, c64 = c64)
    if (is.null(c64)) {
        L4m <- log(exp(ag$x["a4"]) + 1)
    }
```

```
    else {
        d4 <- samp$dwelltime["204Pb"]
        d6 <- samp$dwelltime["206Pb"]
        L4m <- log(exp(ag$x["a4"]) + 1) + log(c64 * d6/d4)
    }
    L6m <- aLm$x["L6m"]
    L7m <- aLm$x["L7m"]
    LUm <- aLm$x["LUm"]
    den <- exp(L6m - L4m) + exp(L6m) + exp(L7m + L6m) + exp(LUm) +
        1
    p4 <- exp(L6m - L4m)/den
    p6 <- exp(L6m)/den
    p7 <- exp(L7m + L6m)/den
    pU <- 1/den
    pUO <- exp(LUm)/den
    p <- c(p4, p6, p7, pU, pUO)
    rs <- rowSums(samp$counts[, simplex])
    dt <- samp$dwelltime[simplex]
    counts <- matrix(rs, ncol = 1) %*% matrix(p, nrow = 1)
    cps <- sweep(counts, MARGIN = 2, FUN = "/", dt)
    colnames(cps) <- simplex
    cps
  }
```

---

raw_count_ratios

Usage

```
raw_count_ratios(samp)
```

Arguments

```
samp
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (samp)
{
    counts <- samp$counts[, c("206Pb", "207Pb", "238U", "238U16O2")]
    lc <- log(counts)
    nt <- nrow(lc)
    n6 <- counts[, "206Pb"]
    n7 <- counts[, "207Pb"]
    nU <- counts[, "238U"]
```

```
    nUO <- counts[, "238U16O2"]
    L6m <- lc[, "206Pb"] - lc[, "238U"]
    L7m <- lc[, "207Pb"] - lc[, "206Pb"]
    LUm <- lc[, "238U16O2"] - lc[, "238U"]
    rs <- rowSums(counts)
    D <- exp(L6m) + exp(L7m + L6m) + exp(LUm) + 1
    dDdL6m <- exp(L6m) + exp(L7m + L6m)
    dDdL7m <- exp(L7m + L6m)
    dDdLUm <- exp(LUm)
    d2DdL6m2 <- exp(L6m) + exp(L7m + L6m)
    d2DdL7m2 <- exp(L7m + L6m)
    d2DdLUm2 <- exp(LUm)
    d2DdL6mdL7m <- exp(L7m + L6m)
    d2DdL7mdL6m <- exp(L7m + L6m)
    d2DdL6mdLUm <- 0
    d2DdL7mdLUm <- 0
    d2DdLUmdL6m <- 0
    d2DdLUmdL7m <- 0
    d2LLdL6m2 <- (rs/D^2) * (dDdL6m)^2 - (rs/D) * d2DdL6m2
    d2LLdL6mdL7m <- (rs/D^2) * dDdL6m * dDdL7m - (rs/D) * d2DdL6mdL7m
    d2LLdL6mdLUm <- (rs/D^2) * dDdL6m * dDdLUm - (rs/D) * d2DdL6mdLUm
    d2LLdL7m2 <- (rs/D^2) * (dDdL7m)^2 - (rs/D) * d2DdL7m2
    d2LLdL7mdL6m <- (rs/D^2) * dDdL7m * dDdL6m - (rs/D) * d2DdL6mdL7m
    d2LLdL7mdLUm <- (rs/D^2) * dDdL7m * dDdLUm - (rs/D) * d2DdL7mdLUm
    d2LLdLUm2 <- (rs/D^2) * (dDdLUm)^2 - (rs/D) * d2DdLUm2
    d2LLdLUmdL6m <- (rs/D^2) * dDdLUm * dDdL6m - (rs/D) * d2DdLUmdL6m
    d2LLdLUmdL7m <- (rs/D^2) * dDdLUm * dDdL7m - (rs/D) * d2DdLUmdL7m
    out <- list()
    labels <- c("L6m", "L7m", "LUm")
    for (i in 1:nt) {
        out[[i]] <- list()
        out[[i]]$x <- c(L6m[i], L7m[i], LUm[i])
        H <- matrix(0, 3, 3)
        H[1, 1] <- d2LLdL6m2[i]
        H[1, 2] <- d2LLdL6mdL7m[i]
        H[1, 3] <- d2LLdL6mdLUm[i]
        H[2, 1] <- d2LLdL7mdL6m[i]
        H[2, 2] <- d2LLdL7m2[i]
        H[2, 3] <- d2LLdL7mdLUm[i]
        H[3, 1] <- d2LLdLUmdL6m[i]
        H[3, 2] <- d2LLdLUmdL7m[i]
        H[3, 3] <- d2LLdLUm2[i]
        out[[i]]$H <- H
        names(out[[i]]$x) <- labels
        colnames(out[[i]]$H) <- labels
        rownames(out[[i]]$H) <- labels
    }
    out
}
```

---

read_Cameca_asc

Usage

```
read_Cameca_asc(fname)
```

Arguments

```
fname
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (fname)
{
    f <- file(fname)
    open(f)
    out <- list()
    while (length(line <- readLines(f, n = 1, warn = FALSE)) >
        0) {
        if (grepl("ACQUISITION PARAMETERS", line)) {
            block <- readLines(f, n = 12, warn = FALSE)
            ions <- unlist(strsplit(block[2], split = "\t"))[-1]
            out$ions <- gsub(" ", "", ions)
            out$dwelltime <- as.numeric(unlist(strsplit(block[7],
                split = "\t"))[-1])
            detector <- unlist(strsplit(block[12], split = "\t"))[-1]
            out$detector <- gsub(" ", "", detector)
            names(out$dwelltime) <- out$ions
            names(out$detector) <- out$ions
        }
        if (grepl("DETECTOR PARAMETERS", line)) {
            block <- readLines(f, n = 3, warn = FALSE)
            detectors <- NULL
            out$yield <- NULL
            out$background <- NULL
            while (TRUE) {
                line <- readLines(f, n = 1, warn = FALSE)
                if (grepl("CORRECTION FACTORS", line)) {
                  break
                }
                else if (nchar(line) > 0) {
                  detectorpars <- gsub(" ", "", unlist(strsplit(line,
                    split = "\t")))
                  detectors <- c(detectors, detectorpars[1])
                  out$yield <- c(out$yield, as.numeric(detectorpars[2]))
                  out$background <- c(out$background, as.numeric(detectorpars[3]))
                }
            }
            names(out$yield) <- detectors
            names(out$background) <- detectors
```

```
        }
        if (grepl("RAW DATA", line)) {
            out$cps <- NULL
            junk <- readLines(f, n = 5, warn = FALSE)
            while ((line <- readLines(f, n = 1, warn = FALSE)) !=
                "") {
                dat <- as.numeric(unlist(strsplit(line, split = "\t")))[-c(1,
                  2)]
                out$cps <- rbind(out$cps, dat)
            }
            colnames(out$cps) <- out$ions
            out$counts <- round(sweep(out$cps, MARGIN = 2, FUN = "*",
                out$dwelltime))
        }
        if (grepl("PRIMARY INTENSITY", line)) {
            out$sbm <- NULL
            junk <- readLines(f, n = 5, warn = FALSE)
            while ((line <- readLines(f, n = 1, warn = FALSE)) !=
                "") {
                dat <- as.numeric(unlist(strsplit(line, split = "\t")))[-c(1,
                  2)]
                out$sbm <- rbind(out$sbm, dat)
            }
            colnames(out$sbm) <- out$ions
        }
        if (grepl("TIMING", line)) {
            out$time <- NULL
            junk <- readLines(f, n = 5, warn = FALSE)
            while (length(line <- readLines(f, n = 1, warn = FALSE)) >
                0) {
                dat <- as.numeric(unlist(strsplit(line, split = "\t")))[-c(1,
                  2)]
                out$time <- rbind(out$time, dat)
            }
            colnames(out$time) <- out$ions
        }
    }
    close(f)
    out
}
```

`read_directory` *Read data directory*

Description

Read all the input files in a data directory

Usage

```
read_directory(dname, instrument = "Cameca", suffix = NULL)
```

Arguments

dname	path to the input directory
instrument	text string with the type of ICP-MS. Currently only 'Cameca'.
suffix	(optional) file extension of the input files.

Value

An object of class simplex, i.e. a list of lists containing the following items: ions, ions, dwelltime, detector, yield, background, cps, counts, sbm, and time.

Examples

```
datadir <- system.file(package="simplex")
dat <- read_directory(datadir,instrument='Cameca',suffix='.asc')
```

---

read_file

Usage

```
read_file(fname, instrument = "Cameca", suffix = ".asc")
```

Arguments

fname

instrument

suffix

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (fname, instrument = "Cameca", suffix = ".asc")
{
    if (instrument == "Cameca" & suffix == ".asc") {
        out <- read_Cameca_asc(fname)
    }
    else if (instrument == "SHRIMP" & suffix == ".op") {
        out <- read_SHRIMP_op(fname)
    }
    out
  }
```

reshuffle

Usage

```
reshuffle(samps, c64 = NULL)
```

Arguments

samps

c64

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (samps, c64 = NULL)
{
    out <- list(counts = list(), cps = list(), time = list(),
        sbm = list(), dwelltime = NULL)
    snames <- names(samps)
    ns <- length(snames)
    ions <- names(samps[[1]]$dwelltime)
    for (i in 1:ns) {
        samp <- samps[[i]]
        for (ion in ions) {
            out$counts[[ion]] <- cbind(out$counts[[ion]], samp$counts[,
                ion])
            out$cps[[ion]] <- cbind(out$cps[[ion]], samp$cps[,
                ion] + 0.5/samp$dwelltime[ion])
            out$time[[ion]] <- cbind(out$time[[ion]], samp$time[,
                ion])
            out$sbm[[ion]] <- cbind(out$sbm[[ion]], samp$sbm[,
                ion])
        }
        out$dwelltime <- rbind(out$dwelltime, samp$dwelltime)
    }
    for (ion in ions) {
        colnames(out$counts[[ion]]) <- snames
        colnames(out$cps[[ion]]) <- snames
        colnames(out$time[[ion]]) <- snames
    }
    out$X <- log(out$cps[["238U16O2"]]) - log(out$cps[["238U"]])
    if (is.null(c64)) {
        out$Y <- log(out$cps[["206Pb"]]) - log(out$cps[["238U"]])
    }
    else {
```

```
        out$Y <- log(out$cps[["206Pb"]] - out$cps[["204Pb"]] *
            c64) - log(out$cps[["238U"]])
    }
    colnames(out$X) <- snames
    colnames(out$Y) <- snames
    rownames(out$dwelltime) <- snames
    out
  }
```

---

subset_samples

Usage

```
subset_samples(dat, prefix = "Plesovice")
```

Arguments

```
dat
prefix
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (dat, prefix = "Plesovice")
{
    snames <- names(dat)
    matches <- grepl(prefix, snames)
    subset(dat, subset = matches)
  }
```

---

yorkfit

Usage

```
yorkfit(XY, omit = NULL)
```

Arguments

```
XY
omit
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (XY, omit = NULL)
{
    if (!is.null(omit)) {
        keep <- !((1:length(XY)) %in% omit)
        XY <- subset(XY, subset = keep)
    }
    snames <- names(XY)
    ns <- length(snames)
    X <- rep(0, ns)
    Y <- rep(0, ns)
    for (i in 1:ns) {
        X[i] <- XY[[i]]$x["X"]
        Y[i] <- XY[[i]]$x["Y"]
    }
    init <- lm(Y ~ X)$coef
    optim(init, misfit_york, method = "BFGS", XY = XY, hessian = TRUE)
  }
```

Index