will\_findChromPeaks\_milliWave <- function (object, param, ...){

.local <- function (object, param, BPPARAM = bpparam(), return.type = "XCMSnExp",

msLevel = 1L)

{

return.type <- match.arg(return.type, c("XCMSnExp",

"list", "xcmsSet"))

startDate <- date()

if (length(msLevel) > 1)

stop("Currently only peak detection in a single MS level is ",

"supported", call. = FALSE)

object\_mslevel <- filterMsLevel(

selectFeatureData(object, fcol = c(MSnbase:::.MSnExpReqFvarLabels, "centroided")),

msLevel. = msLevel

)

if (length(object\_mslevel) == 0)

stop("No MS level ", msLevel, " spectra present to perform ",

"peak detection", call. = FALSE)

centroided <- all(centroided(object\_mslevel))

if (is.na(centroided)) {

suppressWarnings(centroided <- isCentroided(object\_mslevel[[ceiling(length(object\_mslevel)/3)]]))

}

if (is.na(centroided) || !centroided)

warning("Your data appears to be not centroided! CentWave",

" works best on data in centroid mode.")

if (hasAdjustedRtime(object\_mslevel))

fData(object\_mslevel)$retentionTime <- adjustedRtime(object\_mslevel)

object\_mslevel <- lapply(1:length(fileNames(object\_mslevel)),

FUN = filterFile, object = object\_mslevel)

resList <- bplapply(object\_mslevel, FUN = will\_findChromPeaks\_OnDiskMSnExp,

method = "milliWave", param = param, BPPARAM = BPPARAM)

rm(object\_mslevel)

res <- xcms:::.processResultList(resList, getProcHist = return.type ==

"xcmsSet", fnames = fileNames(object))

if (return.type == "list")

return(res$peaks)

object <- xcms:::.peaks\_to\_result(res, object, startDate, param, msLevel)

if (return.type == "xcmsSet")

as(object, "xcmsSet")

else object

}

.local(object, param, ...)

}

will\_findChromPeaks\_OnDiskMSnExp <- function (object, method = "milliWave", param) {

require("xcms", quietly = TRUE, character.only = TRUE)

if (missing(param))

stop("'param' has to be specified!")

will\_findChromPeaks\_Spectrum\_list(x = spectra(object, BPPARAM = SerialParam()),

method = method, param = param, rt = rtime(object))

}

will\_findChromPeaks\_Spectrum\_list <- function (x, method = "milliWave", param, rt) {

method <- match.arg(method, c("centWave", "massifquant", "matchedFilter",

"MSW", "centWaveWithPredIsoROIs", "milliWave"))

method <- paste0("do\_findChromPeaks\_", method)

if (method == "do\_findChromPeaks\_MSW")

method <- "do\_findPeaks\_MSW"

if (method == "do\_findChromPeaks\_matchedFilter") {

x <- lapply(x, function(z) {

if (!length(z@mz)) {

z@mz <- 0

z@intensity <- 0

}

z

})

}

if (missing(param))

stop("'param' has to be specified!")

if (missing(rt))

rt <- unlist(lapply(x, rtime), use.names = FALSE)

if (is.unsorted(rt))

stop("Spectra are not ordered by retention time!")

mzs <- lapply(x, mz)

vals\_per\_spect <- lengths(mzs, FALSE)

procDat <- date()

res <- do.call(method, args = c(list(mz = unlist(mzs, use.names = FALSE),

int = unlist(lapply(x, intensity),

use.names = FALSE),

valsPerSpect = vals\_per\_spect,

scantime = rt),

as(param, "list")))

rm(mzs)

rm(x)

rm(rt)

gc()

list(peaks = res, date = procDat)

}

do\_findChromPeaks\_milliWave <- function (mz, int, scantime, valsPerSpect,

ppm = 25, peakwidth = c(20, 50),

snthresh = 10, prefilter = c(3, 100),

mzCenterFun = "wMean", integrate = 1,

mzdiff = -0.001, fitgauss = FALSE,

noise = 0, verboseColumns = FALSE,

roiList = list(), firstBaselineCheck = TRUE,

roiScales = NULL){

.milliWave(mz = mz, int = int, scantime = scantime,

valsPerSpect = valsPerSpect, ppm = ppm, peakwidth = peakwidth,

snthresh = snthresh, prefilter = prefilter, mzCenterFun = mzCenterFun,

integrate = integrate, mzdiff = mzdiff, fitgauss = fitgauss,

noise = noise, verboseColumns = verboseColumns, roiList = roiList,

firstBaselineCheck = firstBaselineCheck, roiScales = roiScales)

}

.milliWave <- function (mz, int, scantime, valsPerSpect, ppm = 25,

peakwidth = c(20, 50), snthresh = 10,

prefilter = c(3, 100), mzCenterFun = "wMean",

integrate = 1, mzdiff = -0.001, fitgauss = FALSE,

noise = 0, verboseColumns = FALSE,

roiList = list(), firstBaselineCheck = TRUE,

roiScales = NULL) {

if (missing(mz) | missing(int) | missing(scantime) | missing(valsPerSpect))

stop("Arguments 'mz', 'int', 'scantime' and 'valsPerSpect'",

" are required!")

if (length(mz) != length(int) | length(valsPerSpect) != length(scantime) |

length(mz) != sum(valsPerSpect))

stop("Lengths of 'mz', 'int' and of 'scantime','valsPerSpect'",

" have to match. Also, 'length(mz)' should be equal to",

" 'sum(valsPerSpect)'.")

scanindex <- xcms:::valueCount2ScanIndex(valsPerSpect)

if (!is.double(mz))

mz <- as.double(mz)

if (!is.double(int))

int <- as.double(int)

mzCenterFun <- "weighted.mean"

if (!exists(mzCenterFun, mode = "function"))

stop("Function '", mzCenterFun, "' not defined !")

if (!is.logical(firstBaselineCheck))

stop("Parameter 'firstBaselineCheck' should be logical!")

if (length(firstBaselineCheck) != 1)

stop("Parameter 'firstBaselineCheck' should be a single logical !")

if (length(roiScales) > 0)

if (length(roiScales) != length(roiList) | !is.numeric(roiScales))

stop("If provided, parameter 'roiScales' has to be a numeric with",

" length equal to the length of 'roiList'!")

basenames <- c("mz", "mzmin", "mzmax",

"rt", "rtmin", "rtmax", "into",

"intb", "maxo", "sn")

verbosenames <- c("egauss", "mu", "sigma",

"h", "f", "dppm", "scale", "scpos",

"scmin", "scmax", "lmin", "lmax")

scalerange <- round((peakwidth/mean(diff(scantime)))/2)

if (length(z <- which(scalerange == 0)))

scalerange <- scalerange[-z]

if (length(scalerange) < 1) {

warning("No scales? Please check peak width!")

if (verboseColumns) {

nopeaks <- matrix(nrow = 0, ncol = length(basenames) +

length(verbosenames))

colnames(nopeaks) <- c(basenames, verbosenames)

}

else {

nopeaks <- matrix(nrow = 0, ncol = length(basenames))

colnames(nopeaks) <- c(basenames)

}

return(invisible(nopeaks))

}

if (length(scalerange) > 1)

scales <- seq(from = scalerange[1], to = scalerange[2],

by = 2)

else scales <- scalerange

minPeakWidth <- scales[1]

noiserange <- c(minPeakWidth \* 3, max(scales) \* 3)

maxGaussOverlap <- 0.5

minPtsAboveBaseLine <- max(4, minPeakWidth - 2)

minCentroids <- minPtsAboveBaseLine

scRangeTol <- maxDescOutlier <- floor(minPeakWidth/2)

scanrange <- c(1, length(scantime))

if (length(roiList) == 0) {

message("Detecting mass traces at ", ppm, " ppm ... ",

appendLF = FALSE)

withRestarts(tryCatch({

tmp <- capture.output(roiList <- .Call("xcms:::findmzROI",

mz, int, scanindex, as.double(c(0, 0)), as.integer(scanrange),

as.integer(length(scantime)), as.double(ppm \* 1e-06),

as.integer(minCentroids), as.integer(prefilter),

as.integer(noise), PACKAGE = "xcms"))

}, error = function(e) {

if (grepl("m/z sort assumption violated !",

e$message)) {

invokeRestart("fixSort")

}

else {

simpleError(e)

}

}), fixSort = function() {

splitF <- Rle(1:length(valsPerSpect), valsPerSpect)

mzl <- as.list(S4Vectors::split(mz, f = splitF))

oidx <- lapply(mzl, order)

mz <<- unlist(mapply(mzl, oidx, FUN = function(y,

z) {

return(y[z])

}, SIMPLIFY = FALSE, USE.NAMES = FALSE), use.names = FALSE)

int <<- unlist(mapply(as.list(split(int, f = splitF)),

oidx, FUN = function(y, z) {

return(y[z])

}, SIMPLIFY = FALSE, USE.NAMES = FALSE), use.names = FALSE)

rm(mzl)

rm(splitF)

tmp <- capture.output(roiList <<- .Call("findmzROI",

mz, int, scanindex, as.double(c(0, 0)),

as.integer(scanrange),

as.integer(length(scantime)),

as.double(ppm \* 1e-06),

as.integer(minCentroids),

as.integer(prefilter),

as.integer(noise), PACKAGE = "xcms"))

})

message("OK")

if (length(roiList) == 0) {

warning("No ROIs found! \n")

if (verboseColumns) {

nopeaks <- matrix(nrow = 0, ncol = length(basenames) +

length(verbosenames))

colnames(nopeaks) <- c(basenames, verbosenames)

}

else {

nopeaks <- matrix(nrow = 0, ncol = length(basenames))

colnames(nopeaks) <- c(basenames)

}

return(invisible(nopeaks))

}

}

peaklist <- list()

Nscantime <- length(scantime)

lf <- length(roiList)

message("Detecting chromatographic peaks in ", length(roiList),

" regions of interest ...", appendLF = FALSE)

for (f in 1:lf) {

feat <- roiList[[f]]

N <- feat$scmax - feat$scmin + 1

peaks <- peakinfo <- NULL

mzrange <- c(feat$mzmin, feat$mzmax)

mzrange\_ROI <- mzrange

sccenter <- feat$scmin[1] + floor(N/2) - 1

scrange <- c(feat$scmin, feat$scmax)

sr <- c(max(scanrange[1], scrange[1] - max(noiserange)),

min(scanrange[2], scrange[2] + max(noiserange)))

eic <- .Call("getEIC", mz, int, scanindex, as.double(mzrange),

as.integer(sr), as.integer(length(scanindex)), PACKAGE = "xcms")

d <- eic$intensity

td <- sr[1]:sr[2]

scan.range <- c(sr[1], sr[2])

idxs <- which(eic$scan %in% seq(scrange[1], scrange[2]))

mzROI.EIC <- list(scan = eic$scan[idxs], intensity = eic$intensity[idxs])

omz <- .Call("getMZ", mz, int, scanindex, as.double(mzrange),

as.integer(scrange), as.integer(length(scantime)),

PACKAGE = "xcms")

if (all(omz == 0)) {

warning("centWave: no peaks found in ROI.")

next

}

od <- mzROI.EIC$intensity

otd <- mzROI.EIC$scan

if (all(od == 0)) {

warning("centWave: no peaks found in ROI.")

next

}

ftd <- max(td[1], scrange[1] - scRangeTol):min(td[length(td)],

scrange[2] + scRangeTol)

fd <- d[match(ftd, td)]

if (N >= 10 \* minPeakWidth) {

noised <- .Call("getEIC", mz, int, scanindex,

as.double(mzrange), as.integer(scanrange), as.integer(length(scanindex)),

PACKAGE = "xcms")$intensity

}

else {

noised <- d

}

noise <- estimateChromNoise(noised, trim = 0.05, minPts = 3 \* minPeakWidth)

if (firstBaselineCheck & !continuousPtsAboveThreshold(fd, threshold = noise,

num = minPtsAboveBaseLine))

next

lnoise <- getLocalNoiseEstimate(d, td, ftd, noiserange,

Nscantime, threshold = noise,

num = minPtsAboveBaseLine)

baseline <- max(1, min(lnoise[1], noise))

sdnoise <- max(1, lnoise[2])

sdthr <- sdnoise \* snthresh

if (!(any(fd - baseline >= sdthr)))

next

wCoefs <- MSW.cwt(d, scales = scales, wavelet = "mexh")

if (!(!is.null(dim(wCoefs)) && any(wCoefs - baseline >= sdthr)))

next

if (td[length(td)] == Nscantime)

wCoefs[nrow(wCoefs), ] <- wCoefs[nrow(wCoefs) - 1,

] \* 0.99

localMax <- MSW.getLocalMaximumCWT(wCoefs)

rL <- MSW.getRidge(localMax)

wpeaks <- sapply(rL, function(x) {

w <- min(1:length(x), ncol(wCoefs))

any(wCoefs[x, w] - baseline >= sdthr)

})

if (any(wpeaks)) {

wpeaksidx <- which(wpeaks)

for (p in 1:length(wpeaksidx)) {

opp <- rL[[wpeaksidx[p]]]

pp <- unique(opp)

if (length(pp) >= 1) {

dv <- td[pp] %in% ftd

if (any(dv)) {

if (any(d[pp[dv]] - baseline >= sdthr)) {

if (length(roiScales) > 0) {

best.scale.nr <- which(scales == roiScales[[f]])

if (best.scale.nr > length(opp))

best.scale.nr <- length(opp)

}

else {

inti <- numeric(length(opp))

irange <- rep(ceiling(scales[1]/2), length(opp))

for (k in 1:length(opp)) {

kpos <- opp[k]

r1 <- ifelse(kpos - irange[k] > 1,

kpos - irange[k], 1)

r2 <- ifelse(kpos + irange[k] < length(d),

kpos + irange[k], length(d))

inti[k] <- sum(d[r1:r2])

}

maxpi <- which.max(inti)

if (length(maxpi) > 1) {

m <- wCoefs[opp[maxpi], maxpi]

bestcol <- which(m == max(m), arr.ind = TRUE)[2]

best.scale.nr <- maxpi[bestcol]

}

else best.scale.nr <- maxpi

}

best.scale <- scales[best.scale.nr]

best.scale.pos <- opp[best.scale.nr]

pprange <- min(pp):max(pp)

lwpos <- max(1, best.scale.pos - best.scale)

rwpos <- min(best.scale.pos + best.scale,

length(td))

p1 <- match(td[lwpos], otd)[1]

p2 <- match(td[rwpos], otd)

p2 <- p2[length(p2)]

if (is.na(p1))

p1 <- 1

if (is.na(p2))

p2 <- N

mz.value <- omz[p1:p2]

mz.int <- od[p1:p2]

maxint <- max(mz.int)

mzorig <- mz.value

mz.value <- mz.value[mz.int > 0]

mz.int <- mz.int[mz.int > 0]

if (length(mz.value) == 0)

next

mzrange <- range(mz.value)

mzmean <- do.call(mzCenterFun, list(mz = mz.value,

intensity = mz.int))

dppm <- NA

if (verboseColumns) {

if (length(mz.value) >= (minCentroids +

1)) {

dppm <- round(min(running(abs(diff(mz.value))/(mzrange[2] \* 1e-06),

fun = max, width = minCentroids)))

}

else {

dppm <- round((mzrange[2] - mzrange[1])/(mzrange[2] \*

1e-06))

}

}

peaks <- rbind(peaks, c(mzmean, mzrange,

NA, NA, NA, NA, NA, maxint,

round((maxint - baseline)/sdnoise),

NA, NA, NA, NA,

f, dppm, best.scale, td[best.scale.pos],

td[lwpos], td[rwpos], NA, NA))

peakinfo <- rbind(peakinfo, c(best.scale,

best.scale.nr, best.scale.pos, lwpos,

rwpos))

}

}

}

}

}

if (!is.null(peaks)) {

colnames(peaks) <- c(basenames, verbosenames)

colnames(peakinfo) <- c("scale", "scaleNr",

"scpos", "scmin", "scmax")

for (p in 1:dim(peaks)[1]) {

mzr <- peaks[p, c("mzmin", "mzmax")]

if (any(mzr != mzrange\_ROI)) {

eic <- .Call("getEIC", mz, int, scanindex,

as.double(mzr), as.integer(sr), as.integer(length(scanindex)),

PACKAGE = "xcms")

current\_ints <- eic$intensity

mzrange\_ROI <- c(0, 0)

}

else {

current\_ints <- d

}

if (integrate == 1) {

lm <- descendMin(wCoefs[, peakinfo[p, "scaleNr"]],

istart = peakinfo[p, "scpos"])

gap <- all(current\_ints[lm[1]:lm[2]] == 0)

if ((lm[1] == lm[2]) || gap)

lm <- descendMinTol(current\_ints, startpos = c(peakinfo[p,

"scmin"], peakinfo[p, "scmax"]),

maxDescOutlier)

}

else {

lm <- descendMinTol(current\_ints, startpos = c(peakinfo[p,

"scmin"], peakinfo[p, "scmax"]),

maxDescOutlier)

}

lm <- .narrow\_rt\_boundaries(lm, d)

lm\_seq <- lm[1]:lm[2]

pd <- current\_ints[lm\_seq]

peakrange <- td[lm]

peaks[p, "rtmin"] <- scantime[peakrange[1]]

peaks[p, "rtmax"] <- scantime[peakrange[2]]

peaks[p, "maxo"] <- max(pd)

pwid <- (scantime[peakrange[2]] - scantime[peakrange[1]])/(peakrange[2] -

peakrange[1])

if (is.na(pwid))

pwid <- 1

peaks[p, "into"] <- pwid \* sum(pd)

db <- pd - baseline

peaks[p, "intb"] <- pwid \* sum(db[db >

0])

peaks[p, "lmin"] <- lm[1]

peaks[p, "lmax"] <- lm[2]

if (fitgauss) {

td\_lm <- td[lm\_seq]

md <- max(pd)

d1 <- pd/md

pgauss <- fitGauss(td\_lm, pd, pgauss = list(mu = peaks[p,

"scpos"], sigma = peaks[p, "scmax"] -

peaks[p, "scmin"], h = peaks[p, "maxo"]))

rtime <- peaks[p, "scpos"]

if (!any(is.na(pgauss)) && all(pgauss > 0)) {

gtime <- td[match(round(pgauss$mu), td)]

if (!is.na(gtime)) {

rtime <- gtime

peaks[p, "mu"] <- pgauss$mu

peaks[p, "sigma"] <- pgauss$sigma

peaks[p, "h"] <- pgauss$h

peaks[p, "egauss"] <- sqrt((1/length(td\_lm)) \*

sum(((d1 - gauss(td\_lm, pgauss$h/md,

pgauss$mu, pgauss$sigma))^2)))

}

}

peaks[p, "rt"] <- scantime[rtime]

if (peaks[p, "rt"] < peaks[p, "rtmin"])

peaks[p, "rt"] <- scantime[peaks[p,

"scpos"]]

}

else peaks[p, "rt"] <- scantime[peaks[p,

"scpos"]]

}

peaks <- joinOverlappingPeaks(td, d, otd, omz, od,

scantime, scan.range, peaks, maxGaussOverlap,

mzCenterFun = mzCenterFun)

}

if (!is.null(peaks)) {

peaklist[[length(peaklist) + 1]] <- peaks

}

}

if (length(peaklist) == 0) {

warning("No peaks found!")

if (verboseColumns) {

nopeaks <- matrix(nrow = 0, ncol = length(basenames) +

length(verbosenames))

colnames(nopeaks) <- c(basenames, verbosenames)

}

else {

nopeaks <- matrix(nrow = 0, ncol = length(basenames))

colnames(nopeaks) <- c(basenames)

}

message(" FAIL: none found!")

return(nopeaks)

}

p <- do.call(rbind, peaklist)

if (!verboseColumns)

p <- p[, basenames, drop = FALSE]

return(p)

uorder <- order(p[, "into"], decreasing = TRUE)

pm <- as.matrix(p[, c("mzmin", "mzmax", "rtmin",

"rtmax"), drop = FALSE])

uindex <- rectUnique(pm, uorder, mzdiff, ydiff = -1e-05)

pr <- p[uindex, , drop = FALSE]

message(" OK: ", nrow(pr), " found.")

return(pr)

}